NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN OF MULTIPLE-VALUED
PROGRAMMABLE LOGIC ARRAYS

by

Yong Ha Ko

December 1988

Thesis Advisor: Jon T. Butler

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) DESIGN OF MULTIPLE-VALUED PROGRAMMABLE LOGIC ARRAYS

12 PERSONAL AUTHOR(S) Ko, Yong H.

13a TYPE OF REPORT Master's Thesis	13b TIME COVERED FROM _____ TO _____	14 DATE OF REPORT (Year, Month, Day) 1988 December	15 PAGE COUNT 70

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Multiple-Valued Logic Function, Programmable Logic Array, Circuit Generation, Simulation

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The goal of this thesis is the development of a programmable logic array(PLA) that accepts multiple-valued inputs and produces multiple-valued outputs. The PLA is implemented in CMOS and multiple levels are encoded as current. It is programmed by choosing transistor geometries which control the current level at which the PLA reacts to inputs. An example of a 4-valued PLA is shown. As part of this research, a C program was written that produces a PLA layout.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Jon T. Butler	22b TELEPHONE (Include Area Code) 408-646-3299 22c OFFICE SYMBOL 62Bu

DD Form 1473, JUN 86 Previous editions are obsolete
S/N 0102-LF-014-6603

Approved for public release; distribution is unlimited

Design of Multiple-Valued Programmable Logic Arrays

by

Yong Ha Ko
Major, Republic of Korea Air Force
B.S., Air Force Academy, CheongJu, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1988

ABSTRACT

The goal of this thesis is the development of a programmable logic array (PLA) that accepts multiple-valued inputs and produces multiple valued outputs. The PLA is implemented in CMOS and multiple levels are encoded as current. It is programmed by choosing transistor geometries which control the current level at which the PLA reacts to inputs. An example of a 4-valued PLA is shown. As part of this research, a C program was written that produces a PLA layout.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I wish to thank Professor Jon T. Butler and Chyan Yang for their support and guidance in this endeavor.

Mostly, I would like to acknowledge my wife Jeong Phil, son Jun Seong and Hye Seong without whose patience this project could not have been completed.

I. INTRODUCTION

The increasing demand for speed and performance in modern information processing systems clearly points to the need for super chips with significant computation power. Low-cost, high-density, fast VLSI devices are essential to make super-computing practical in terms of volume, speed and cost. Multiple-valued LSI and VLSI have a potential advantage that they provide a means of increasing data processing capability per unit area. Multiple-valued logic (MVL) circuits allow interconnections to carry more information, thus reducing chip area. Multiple-valued logic also stands as a solution to the pin-out problem, where a limit on the number of pins in IC packages has limited information flow between packages.

For example, a 32x32-bit SD (signed-digit) multiplier chip with multiple-valued bidirectional current-mode logic circuits has been developed by Japanese researchers [Ref. 1]. Compared to the fastest binary multiplier, the multiple-valued multiplier is superior in terms of power dissipation, effective multiplier size, number of transistors and number of interconnections. The multiply time is almost the same (Refer to comparison table, p. 54 in [Ref. 2]).

Currently, no multiple-valued PLA has been implemented in current mode CMOS technology while an MVL CCD-PLA [Ref. 3] has been implemented and PLAs with decoders [Ref. 4] have been proposed. This study is the first approach to a multiple-valued PLA that is implemented by current-mode CMOS technology.

The main goal of this research is to design a multiple-valued PLA cell with current-mode CMOS technology. The multiple-valued logic function is parsed into the primary elements which correspond to the basic cells. Using those cells, a PLA circuit layout generation program (mvpla) has been written in C language that produces layouts suitable for implementation in IC fabrication facilities. Sample MVL functions are generated and simulated. Because of multilevel inputs and outputs, it is more convenient to verify the circuit with analog signals than with discrete signals. Simulation of the final design is done by the analog level simulator,

which is practical with a single PLA since there are not enough transistors for computation time to be a limiting factor. However, transistor count limitations preclude the simulation of moderately large PLA's.

This research has been done in conjunction with the design of a CAD tool for multiple-valued programmable logic arrays which will produce the actual layout of the PLA after a given function specified by the user [Ref. 5].

II. THEORETICAL BACKGROUND

A. CURRENT THRESHOLD DETECTION

The following definitions apply and will be used throughout:

V_{DS} = drain-to-source voltage

V_{GS} = gate-to-source voltage

V_T = threshold voltage

V_{SW} = switching voltage

V_I = input voltage

I_I = input current

V_O = output voltage

I_O = output current

I_{SW} = switching current

I_{DS} = drain-to-source current

W = channel width of MOSFET

L = channel length of MOSFET

G = geometry ratio(W/L) of MOSFET

β = MOS transistor gain factor

M_i = i-th MOS device name in the circuit description

μ = effective surface mobility of electrons in the channel

ε = permittivity of the gate insulator

t_{ox} = thickness of the gate insulator

R_C = channel resistance

x = variable of multiple-valued logic function

The equations describing the behavior of an ideal nMOS device in three regions are:

- Cut-off region : when the gate-to-source voltage (V_{GS}) is less than the threshold voltage (V_T), no current flows through the transistor.

$$I_D = 0 \qquad \text{if } V_{GS} - V_T < 0. \tag{2.1}$$

- Saturation region : when the gate-to-source voltage is greater than the threshold voltage and the difference (V_{GS}-V_T) is less than the drain-to-source voltage (V_{DS}),

$$I_D = \frac{\beta}{2}(V_{GS} - V_T)^2 \qquad \text{if } 0 < V_{GS} - V_T < V_{DS}. \tag{2.2}$$

- Linear region : when the gate-to-source voltage is greater than the threshold voltage and the difference is greater than the drain-to-source voltage.

$$I_D = \beta\left\{(V_{GS} - V_T)V_{DS} - \frac{V_{DS}^2}{2}\right\} \qquad \text{if } 0 < V_{DS} < V_{GS} - V_T. \tag{2.3}$$

where I_D is the drain-to-source current, V_{GS} is the gate-to-source voltage, V_T is the device threshold, and β is the MOS transistor gain factor. β is dependent on both the process parameters and the device geometry and is given by

$$\beta = \left(\frac{\mu\varepsilon}{t_{ox}}\right)\left(\frac{W}{L}\right). \tag{2.4}$$

The gain factor β thus consists of a process dependent factor ($\mu\varepsilon/t_{ox}$), which depends on all the process terms including doping density, gate oxide thickness, and a geometry dependent term (W/L), which depends on the layout of the device.

An approximate expression for I_D is derived by assuming that the current in the channel saturates (i.e., is constant) and is independent of the applied drain voltage. In saturation [i.e., above $V_{DS} = (V_{GS} - V_T)$], the MOS device behaves like a current source, the current being almost independent of V_{DS}.

In Fig. 2.1, the drain of the nMOS transistor M0 is connected to the gate, which results in saturation when $V_{GS} - V_T \geq 0$ or $V_{DS} > V_T$ (because $V_{DS} = V_{GS}$).

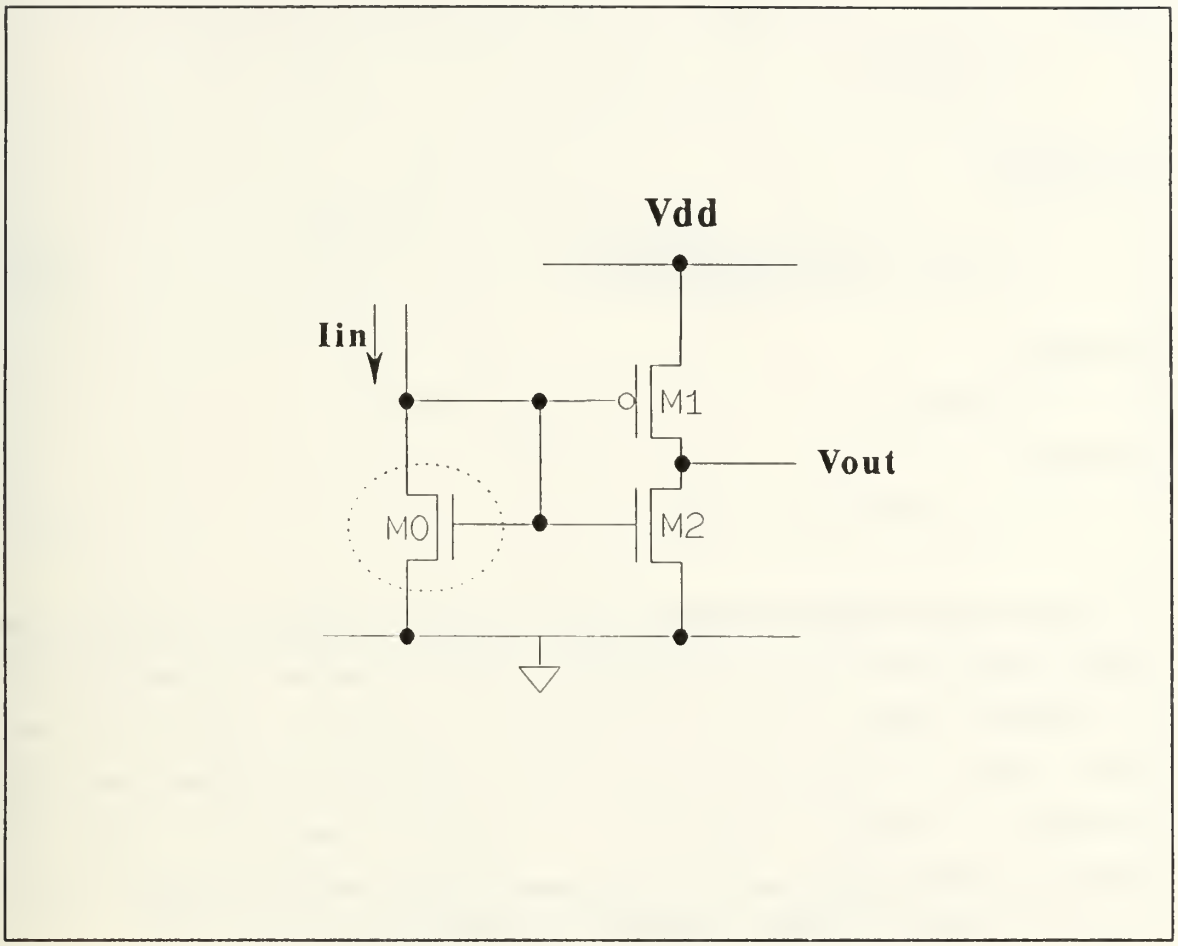


Fig. 2.1 CMOS invertor with the current input

We can rewrite (2.2) as,

$$I_{sw} = C\beta, \tag{2.5}$$

where I_{sw} is the switching current, the constant $C = (V_{sw} - V_T)^2/2$, and V_{sw} is the switching voltage of the CMOS invertor at which V_o changes from the high level to the low level in the CMOS invertor.

When both the pMOS(M1) and the nMOS(M2) (Fig. 2.1) are in saturation, the saturation currents for the two devices are given by

$$I_{DSp} = 0.5\beta_p(V_{sw} - V_{DS} - V_{tp})^2$$

$$I_{DSn} = 0.5\beta_n(V_{sw} - V_{tn})^2$$

with

$$I_{DSp} = -I_{DSn}.$$

This yields

$$V_{SW} = \frac{V_{DS} + V_{tp} + V_{tn}(\beta_n/\beta_p)^{1/2}}{1 + (\beta_n/\beta_p)^{1/2}} \tag{2.6}$$

By setting

$$\beta_n = \beta_p,$$

we obtain

$$V_{SW} = (V_{DS} + V_{tp} + V_{tn})/2. \tag{2.7}$$

The switching voltage V_{SW} depends on the threshold voltages of the PMOS and NMOS transistors M1 and M2, respectively. Since the transistor size does not affect the threshold voltage, the V_{SW}'s are constant on the same IC chip. On the other hand, I_D which is a function of the gain factor β, depends circuit layout, specifically geometries of transistors. Therefore, it is possible to produce the various I_{SW} according to the geometries of nMOS transistor(M0) in Fig. 2.1 while V_{SW} is constant. I_{SW}, written explicitly in terms of transistor geometry, is

$$I_{SW} = C(V_{SW}-V_t)^2(W/L), \tag{2.8}$$

where $C = 0.5\left(\frac{\mu\varepsilon}{t_{ox}}\right)$.

The corresponding I-V transfer characteristic of the circuit in Fig. 2.1 is shown in Fig. 2.2.

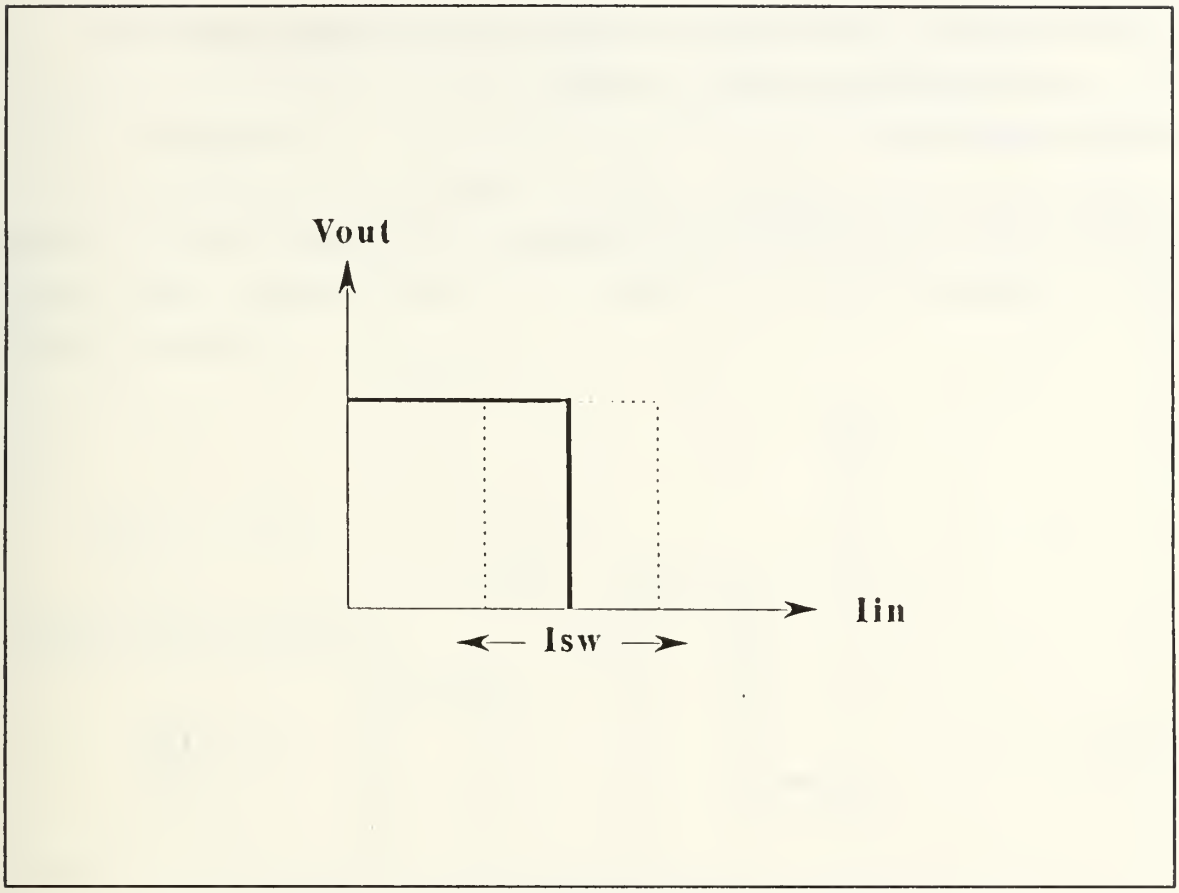


Fig. 2.2 Ideal I-V transfer characteristic of CMOS inverter

B. MULTIPLE-VALUED LOGIC FUNCTION

A general i variable r-valued MVL function can be written as a sum-of-products as follows,

$$f(x_0,x_1,\ldots,x_{i-1}) = \sum_{j=0}^{n-1} R_j \prod_{k=0}^{i-1} x_k(l_{jk},u_{jk}) \tag{2.9}$$

where

$$x(l,u) = \begin{cases} r-1 & \text{if } l \le x \le u, \\ 0 & \text{otherwise.} \end{cases}$$

R_j (the coefficient of the product term) has the property $1 \le R_j \le r-1$, the lower bound l and the upper bound u are integers between 0 and r-1, n is the number of terms, Σ denotes sum the operation (TSUM or truncated summation), and $\prod$ denotes product operation (MIN).

As an example, consider the one-variable four-valued logic function,

$$f(x) = 1x(0,1) + 2x(1,1) + 2x(3,3)$$

shown in Fig. 2.3.

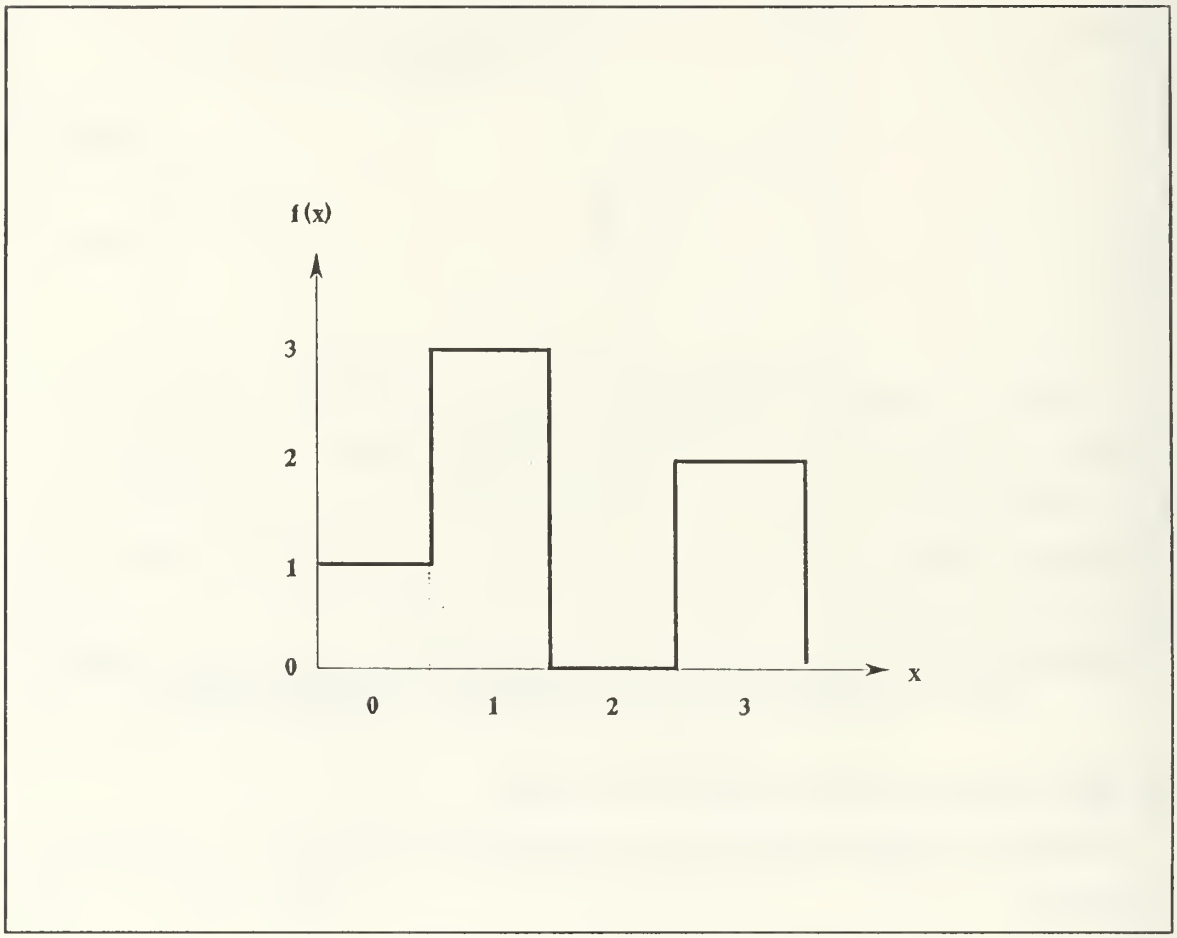


Fig. 2.3 Example function

C. BASIC ELEMENTS OF MULTIPLE-VALUED LOGIC FUNCTION

From Eqs. (2.9), the realization of a multiple-valued function requires three operations

1. truncated sum Σ,
2. MIN $\prod$, and
3. literal x(l,u).

Truncated sum is the most easily implemented operation; it is simply the joining of wires in a wired sum. The MIN operation is more complex in current mode CMOS. However, we can take advantage of the fact that the arguments of the MIN operation are literals and a constant. The literal operation is also more complex. However, it can be realized by recognizing there are two parts to it, a step-up function and a step-down function. We describe the last two operations in the following discussion.

Let

$$\overline{x(l,u)} = \begin{cases} 0 & \text{if } l \le x \le u, \\ r-1 & \text{otherwise.} \end{cases}$$

It follows that product term

$$\prod_{k=0}^{i-1} x_k(l_k,u_k). \tag{2.10}$$

of (2.9) is r-1 if and only if

$$\sum_{k=0}^{i-1} \overline{x_k(l_k,u_k)} \tag{2.11}$$

is 0. Since Σ is so easily realized in current mode CMOS, we choose to realize (2.10) over (2.11). This is done with a wired sum which drives a device that detects the absence or presence of current.

The step functions are obtained by the threshold operation of the current mode CMOS invertor. The elementary operations of a multiple-valued logic function in the current mode CMOS implementation are

- Step-up function (Fig. 2.4a) : x(c,r-1),
- Step-down function (Fig. 2.4b) : x(0,c),
- Truncated sum : TSUM,
- Logic level : R,

where $0 \le c \le r-1$ and R is a coefficient of a product term.

III. DESIGN OF MULTIPLE-VALUED PLA CELLS

In the previous chapter, the elements of the MVL function have been defined. Each element is a basic cell in the circuit layout such as the step-up function generator, the step-down function generator, the column output generator, and the wired adder for the function output. In addition to these cells, an input replicator is needed to replicate the input current to the cells in a row.

A. INPUT REPLICATOR

For a CMOS circuit in voltage-mode operating, fan-out is a straightforward wiring task, but fan-in requires some particular circuitry for each input. In current mode, the reverse is true: fan-in is very simple by wire connection, but fan-out is much more complex, requiring circuitry of the type shown in Fig. 3.1.

There are two different directions of current flow in the current mirror, the positive replicator supplies the forward current (positive direction) to the logic gates while the negative replicator supplies the backward current (negative direction) to the logic gates [Ref. 6]. For the unique direction in current flow from inputs to outputs, neither the positive nor negative replicator are appropriate. It is possible for the input replicator to give the unique direction of current to the inputs of each cells with CMOS current mirror shown as Fig. 3.1. This circuit shows that the negative and positive replicators are connected serially.

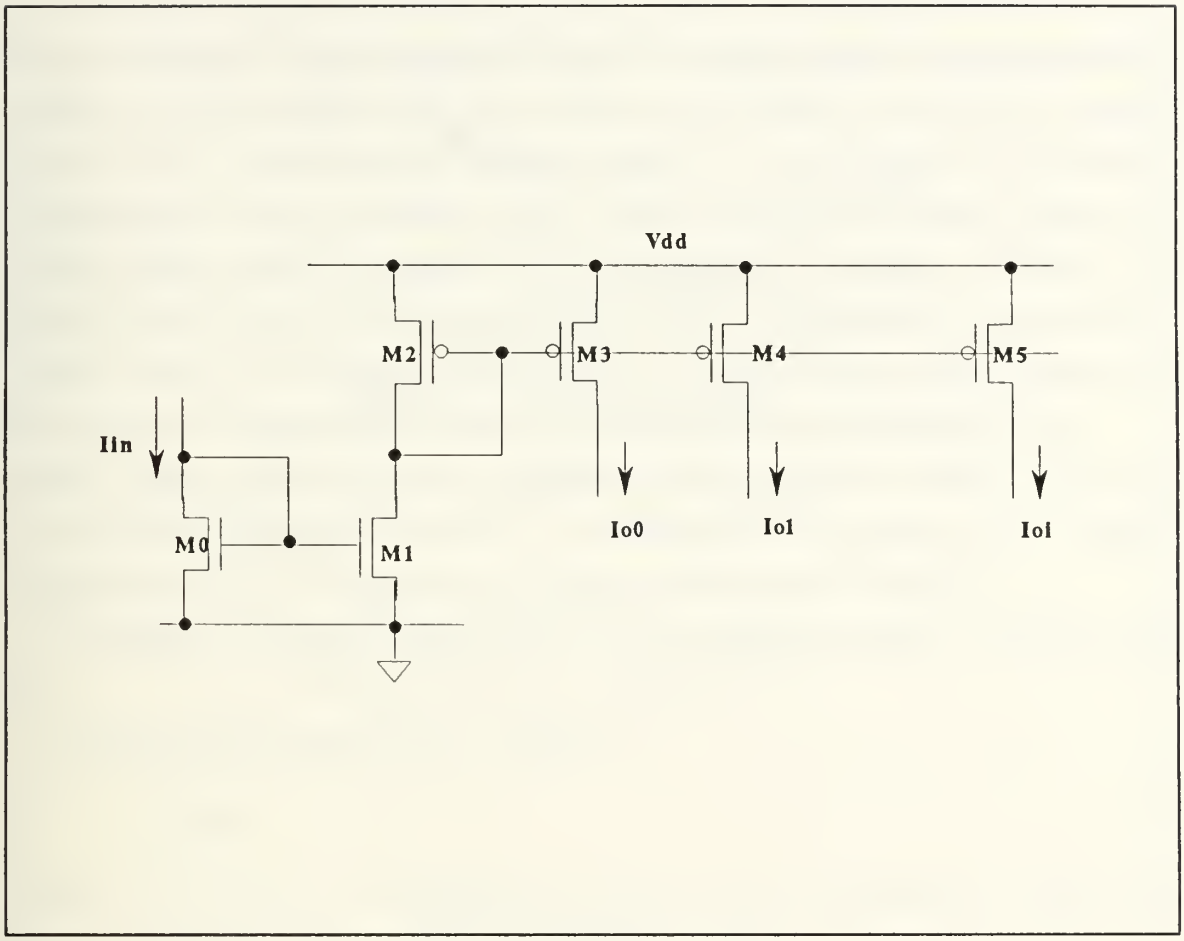
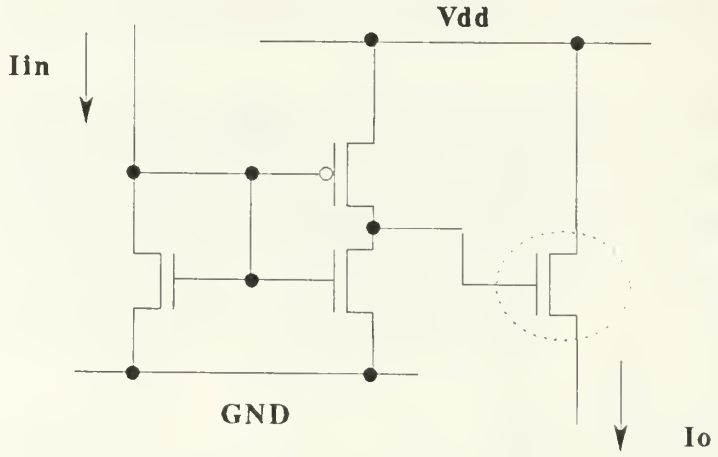


Fig. 3.1 Input replicator

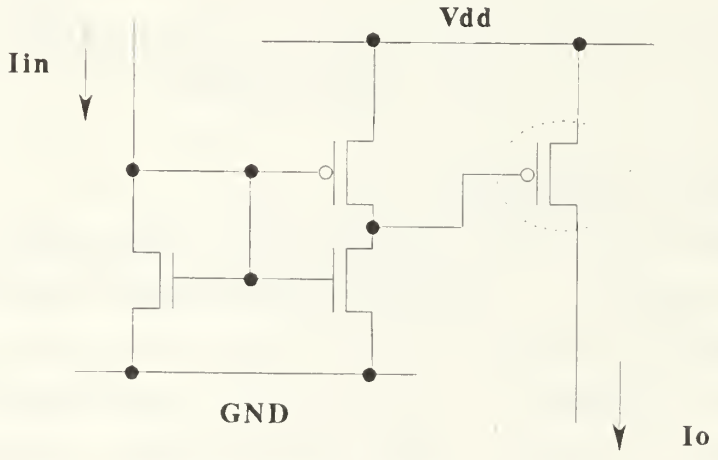
B. THRESHOLD OPERATION

Recall from Chap. II that the literal function generation is composed of two subcircuits, a step-up and a step-down function generator. Both of these subcircuits perform a threshold operation in which the relative value of the input with respect to the threshold is signalled in voltage. However, a current signal is needed.

Specifically, zero current is produced when the input is within prescribed limits. To produce the current output from the voltage signal, an additional MOS device is needed. In the case of the step-up function generation a pMOS transistor is needed; this is a voltage controlled current source M3 in Fig. 3.2.



a) Step-down function generator



a) Step-up function generator

Fig. 3.2 Step function generator

If the input current I_i exceeds I_{sw} of the invertor, V_o switches from low to high with a reasonably sharp transition. When V_o is much higher than the threshold voltage of pMOS transistor M3, the device is in saturation and acts as a constant current source. The current I_o produced depends on the geometry of the pMOS transistor M3. However, the actual value of the current is not important because the column output generator responds only to the presence or absence of current. The output current level of each column (product term) is determined by the number of variable inputs for the upper limit and the I_{sw} of the column output generator for the lower limit. The details for this will be discussed in the description of the column output generator and the simulation of the cells. The circuit of the step-down function generator is similar to that of the step-up generator with a nMOS transistor (of Fig. 3.2a) replacing the pMOS transistor M3 (of Fig. 3.2b).

The ideal DC transfer characteristic of a step function generator is shown in Fig. 3.3.

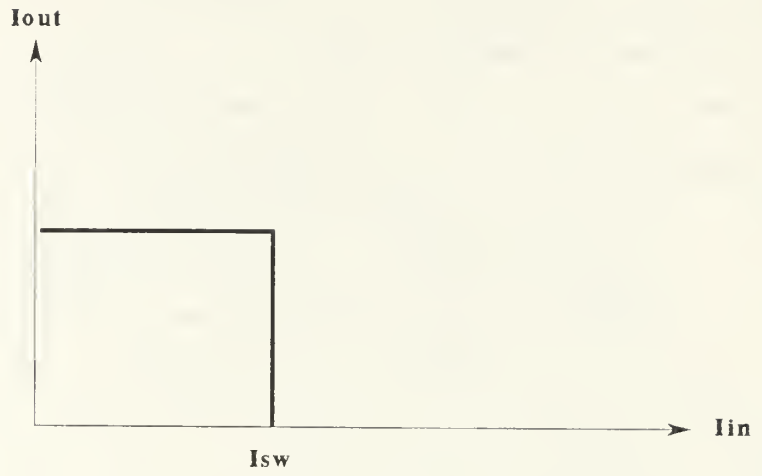
Recall that the literal function is given as

$$\overline{x(l,u)}, \tag{3.1}$$

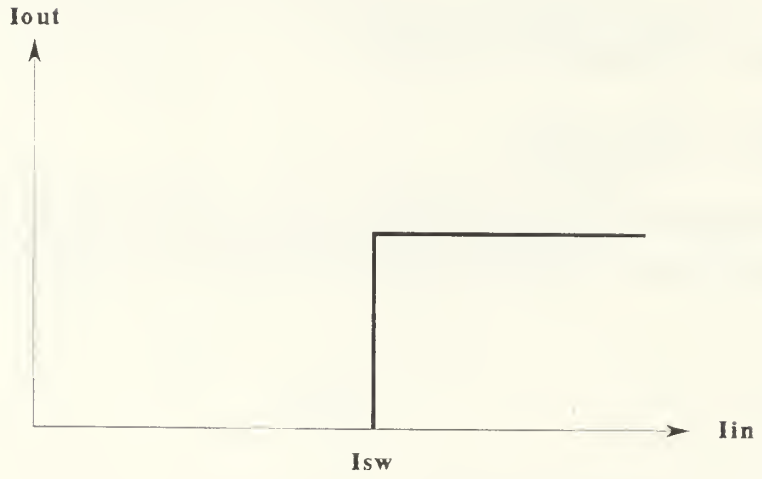
where $0 \leq l \leq u \leq r-1$. This function realized as two subfunctions, a step-up and step-down function. Rewriting Eq. (3.1) yields

$$\overline{x(l,u)} = x(0,l) + x(u,r-1), \tag{3.2}$$

in which the step-up and step-down function in the literal function are expressed separately. Here x(0,l) is a step-down function and x(u,r-1) is a step-up function.



a) Step-down function



b) Step-up function

Fig. 3.3 Ideal DC transfer characteristic of step function generator.

C. COLUMN OUTPUT GENERATOR

The input of the column output generator is the wired-sum of the outputs of the cells. This is simply direct wiring of each output to the input of column output generator. The column output generator produces the coefficient associated with a product term. It is obtained by employing the step-down function generator. As mentioned previously, the input of this cell is either zero or high. When the input is zero, this cell produces the current equal to the product term coefficient, otherwise, the output is zero. The following equation shows the function of this cell. (Refer to Eq. 2.10.)

$$P(p) = R\ \overline{p(0,0)}, \tag{3.3}$$

where P is a product term, p is the wired sum of each cell outputs and R is the coefficient of a product term.

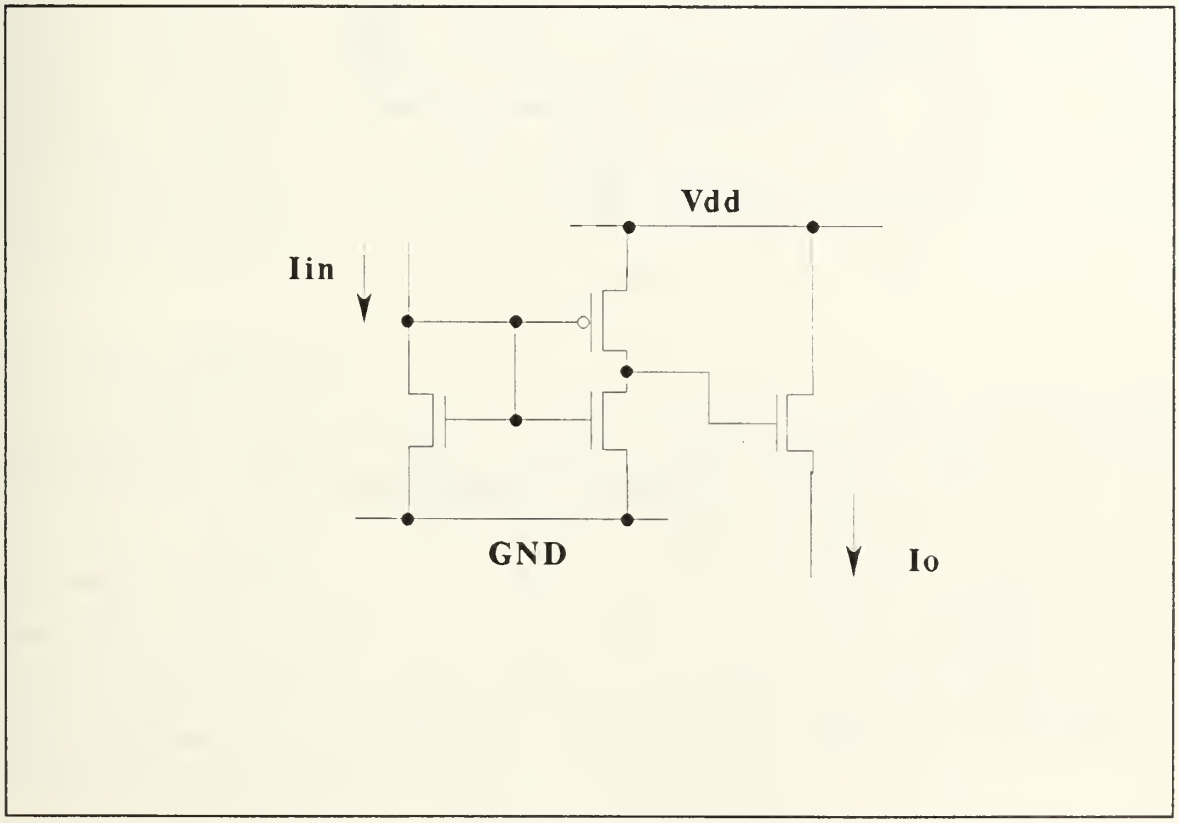


Fig. 3.4 Column output generator

D. FUNCTION OUTPUT

It is interesting to note that the sum operation represented in Eq.(2.11), is not the truncated sum operation used previous papers [Ref. 3, 7, 8]. It is arithmetic addition by Kirchhoff's current law (Fig. 3.5). From a logic design point of view, it is the threshold operation of the function output current that makes it appear as if indeed truncated sum is the logic operation used.

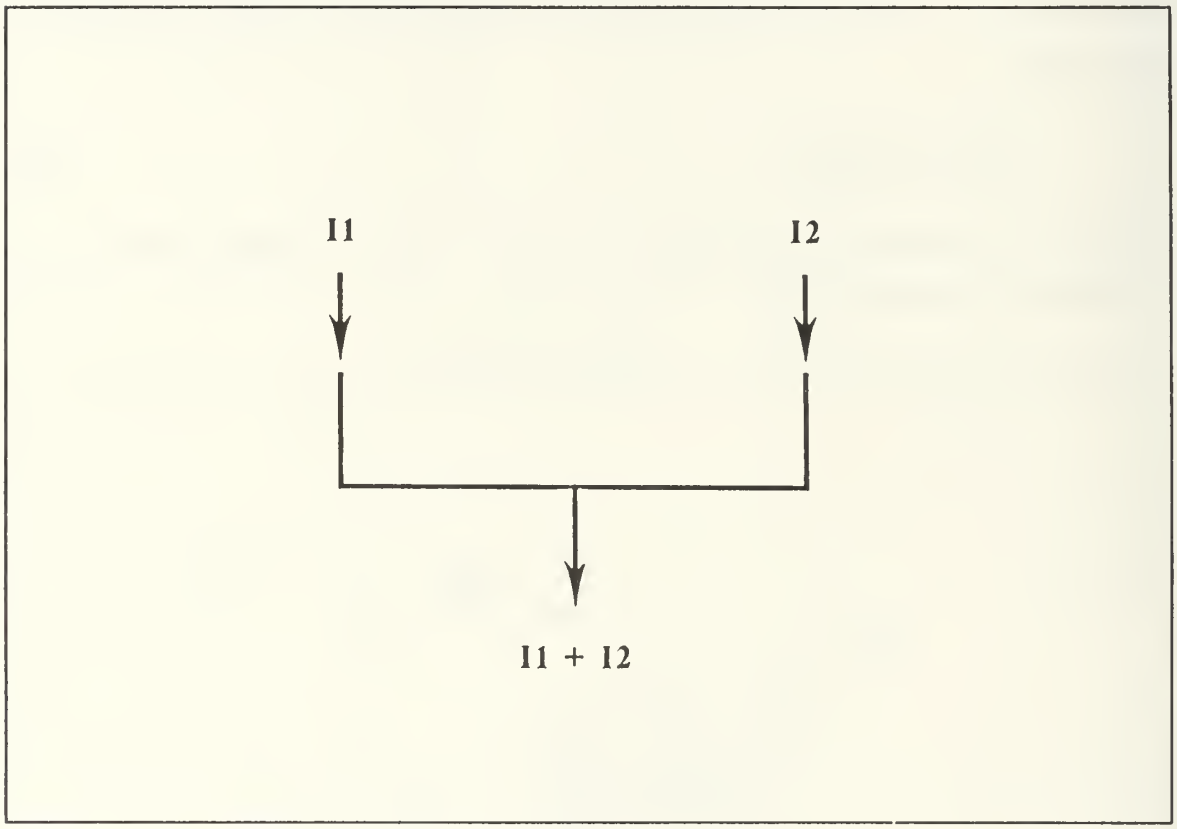


Fig. 3.5 Wired-sum implementation.

IV. SIMULATION OF CELLS

Unlike the switching level simulation in case of binary logic circuits, analog simulation is required for multiple valued logic circuits. The best known program for this is SPICE (simulation program with integrated circuit emphasis). In our work, PSPICE (personal computer version of SPICE) [Ref. 9] was used to simulate the PLA cells, with MODEL parameters supplied by Twente University, Holland. The most important factor is the geometry of devices in these circuits while others are process dependent factors.

Input data files are prepared for the PSPICE program with standard SPICE input format. The outputs of these simulations are given by the DC transfer characteristic. However, a conversion problem on running SPICE occurred when a DC current input was applied to a specific circuitry. The transient analysis was better than DC analysis in this case. Therefore, for the convenience of understanding the results, all circuits are evaluated by transient analysis rather than DC analysis. In our simulations, node 0 is always ground (GND), node 1 is V_{DD}, and node 2 is an input. Dummy voltage sources are connected at the check points with zero voltage to measure current values.

A. INPUT REPLICATOR

The purpose of this subcircuit is to supply current to each cell in the PLA equal to the current at the corresponding input. The input current was applied from 0 μA at 0 nsec to 300 μA at 30 nsec. In the first simulation, all transistors had the same size such as 2 lambda length and 3 lambda width for the minimum size in lambda base design rules (Appendix A). The output of the CMOS replicator did not replicate the input current while increasing the input current as shown Fig. 4.1a. The appropriate replication of input current was obtained by adjusting the size of pMOS transistors as shown Fig. 4.1b.

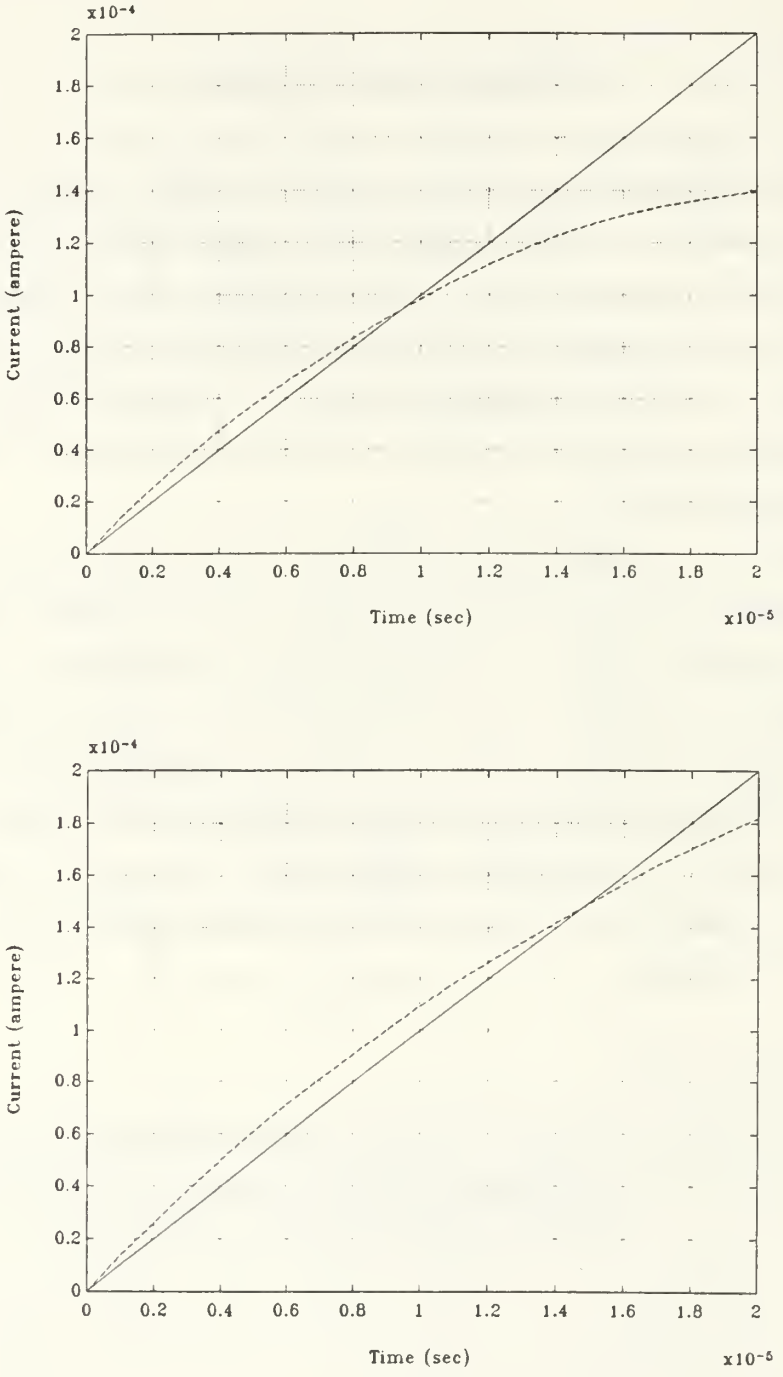


Fig. 4.1 DC transfer characteristic of input replicator.

B. LOGIC LEVEL VS GEOMETRY OF MOSFET

For the threshold operation, the switching points between logic levels are specified by the size of reference transistors (M0's) so that the switching point is on the middle point between logic levels (Fig. 4.4). Thus, the relation between the geometry ratios are obtained as follows.

Define

$$I_{sw} = tI \qquad \text{for } t = 0.5, 1.5, 2.5, \tag{4.1}$$

where I is unit current for logic level one.

Let G_1 the unit geometry ratio for unit current I, and G_{sw} for the threshold, then

$$G_{sw} = tG_1 \qquad \text{for } t = 0.5, 1.5, 2.5. \tag{4.2}$$

For the column output generator, the logic output level I_o is determined by the geometry ratio(W/L) of the transistor, because the saturation current I_D is linearly proportional to β, such that

$$I_o = kI,$$

and,

$$G_o = kG_1 \qquad \text{for } k = 0, 1, 2, 3. \tag{4.3}$$

where G_o is the geometry ratio of output current source.

On the basis of Eq 4.2 and 4.3, Table 4.1 can be obtained for the 4-valued case.

TABLE 4.1 Logic level vs geometry of device,
I = unit current. Unit of W,L = Lambda(=1.5μm)

LEVEL	OUTPUT(W/L)	THRESHOLD(W/L)
0	–	0.5I (3/6)
1	I (3/3)	1.5I (3/2)
2	2I (4/2)	2.5I (5/2)
3	3I (6/2)	

C. STEP-UP OR STEP-DOWN FUNCTION GENERATOR

All CMOS inverters have the same size of transistors. The most important point of this circuit is the switching of the output current in accordance with the different size of transistor (M0) at the input stage (Fig. 3.2). To compare the different switching points at a time during simulation, three similar circuits with different size of M0 were described in a input data file (Appendix B, C).

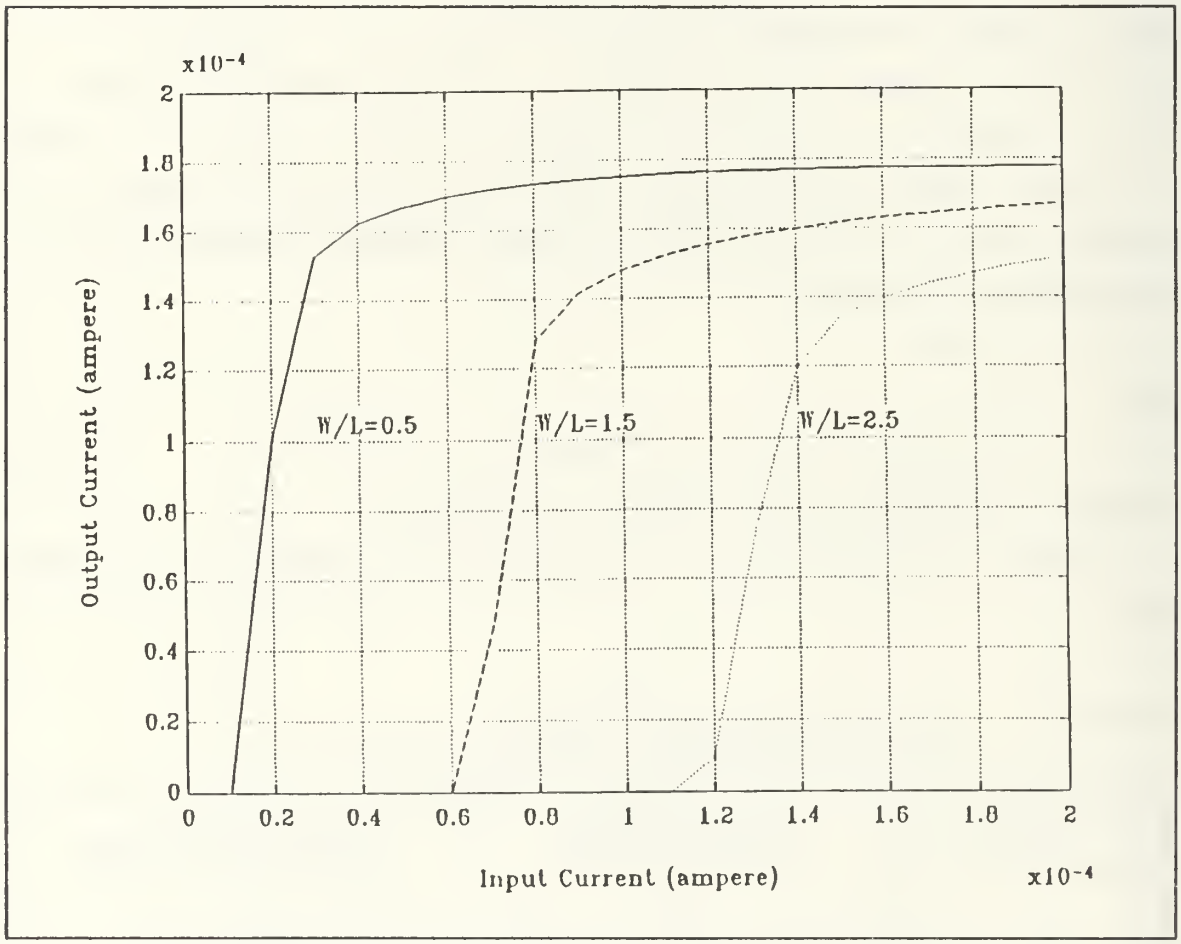


Fig. 4.2 DC transfer characteristic of step-up function generator

The only difference between the step-up and step-down function generator is the type of output current source (M3) (Fig. 3.2) on the invertor output node. The p-type transistor produces the step-up function (Fig. 4.2) and the n-type produces step-down function (Fig. 4.3) at the same switching point.

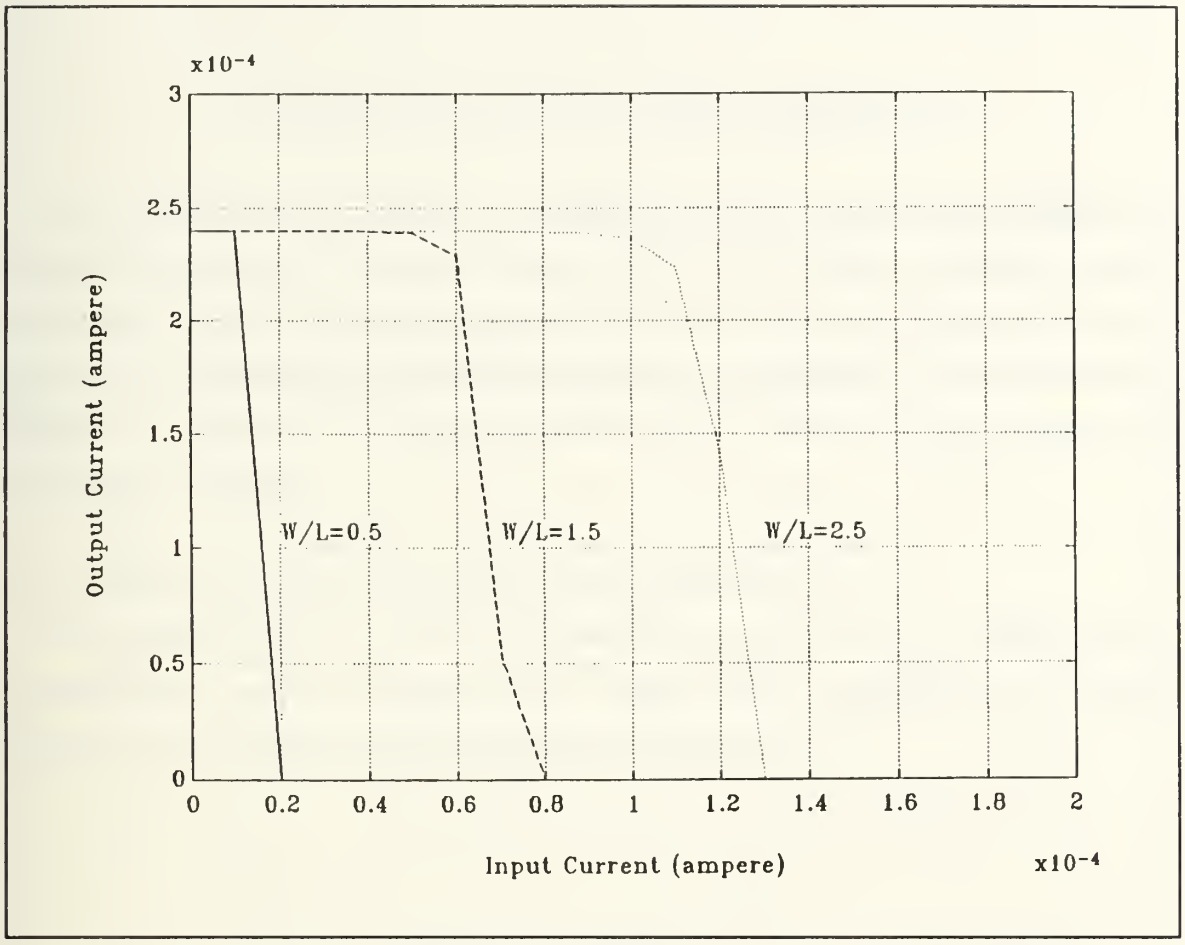


Fig. 4.3 DC transfer characteristic of step-down function generator

D. COLUMN OUTPUT GENERATOR

The simulation result of column output generator is shown in Fig. 4.4. The SPICE data file is attached at Appendix D. The output level is based on Table 4.1.

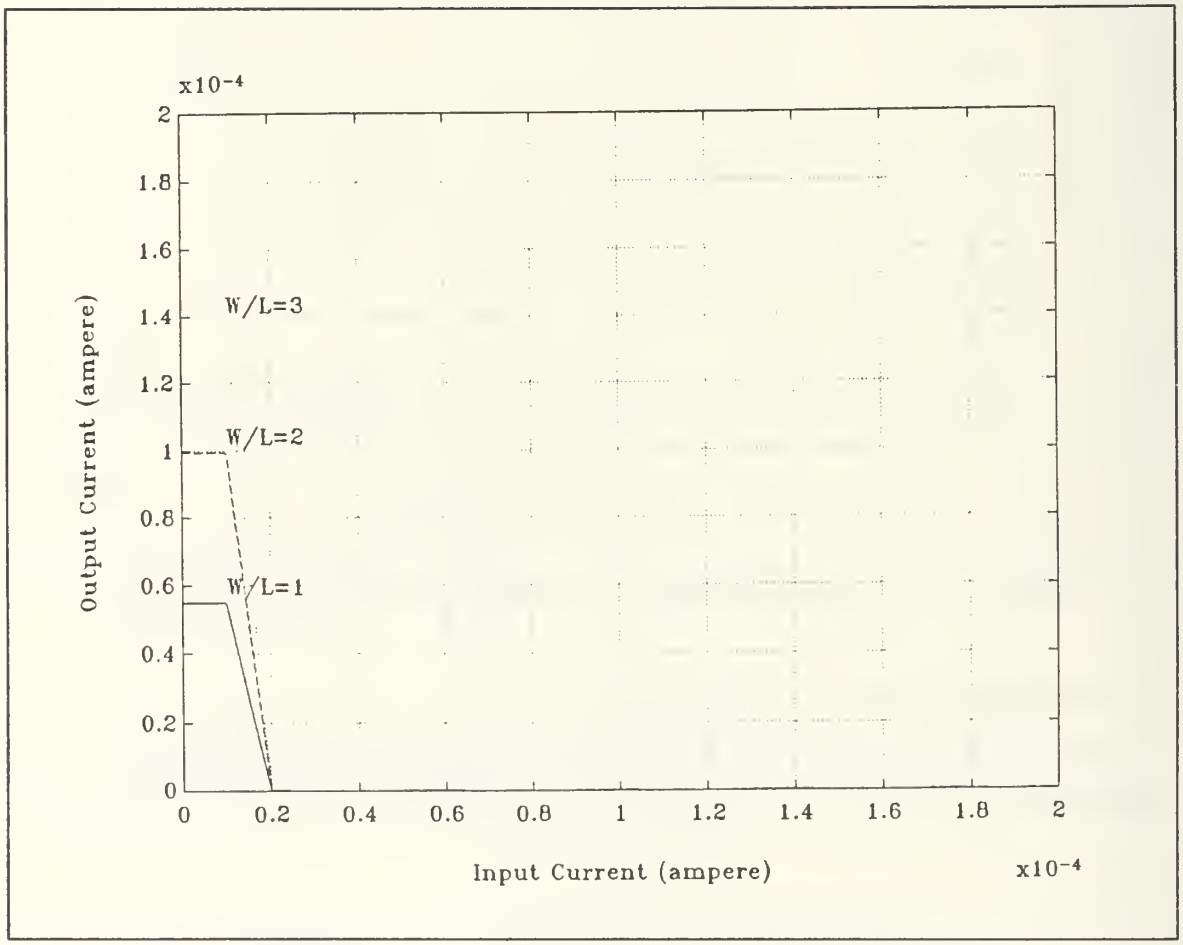


Fig. 4.4 DC transfer characteristic of column output generator

V. GENERATION OF MULTIPLE-VALUED PLA

So far, the elementary cells for the multiple-valued PLA have been designed and simulated individually. The circuit layout of PLA can be built by putting the given cells together. The CFL (Coordinate Free Lap) [Ref. 10] library functions were used to write a program which generates the PLA circuit layout. The output of this program is formed by the MAGIC data file format. The generation program was written in C language.

A. CONCEPTUAL FORM OF MULTIPLE-VALUED PLA

The multiple-valued PLA has two main parts, the column cell corresponding to a product term and the wired sum of the column output corresponding to a function output. The block diagram for these parts are shown Fig. 5.1.

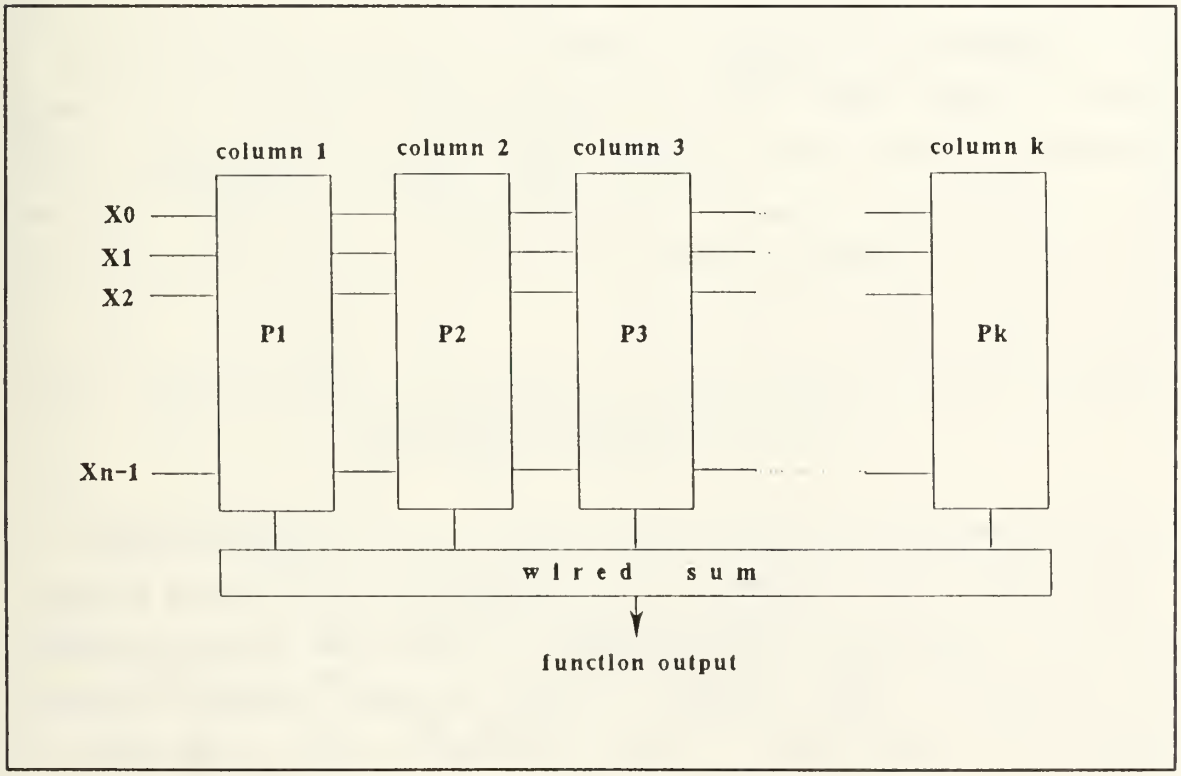


Fig. 5.1 Top level design of PLA

Each column in Fig. 5.1 in turn consists of step function generators, wired-sum together as shown Fig. 5.2. The number of literal function generator cells equals to the number of variables in a product term. The difficulty in this implementation is the various size of a cell, which affects on the regularity in the layout. The optimization of the silicon area should be treated by in further studies.

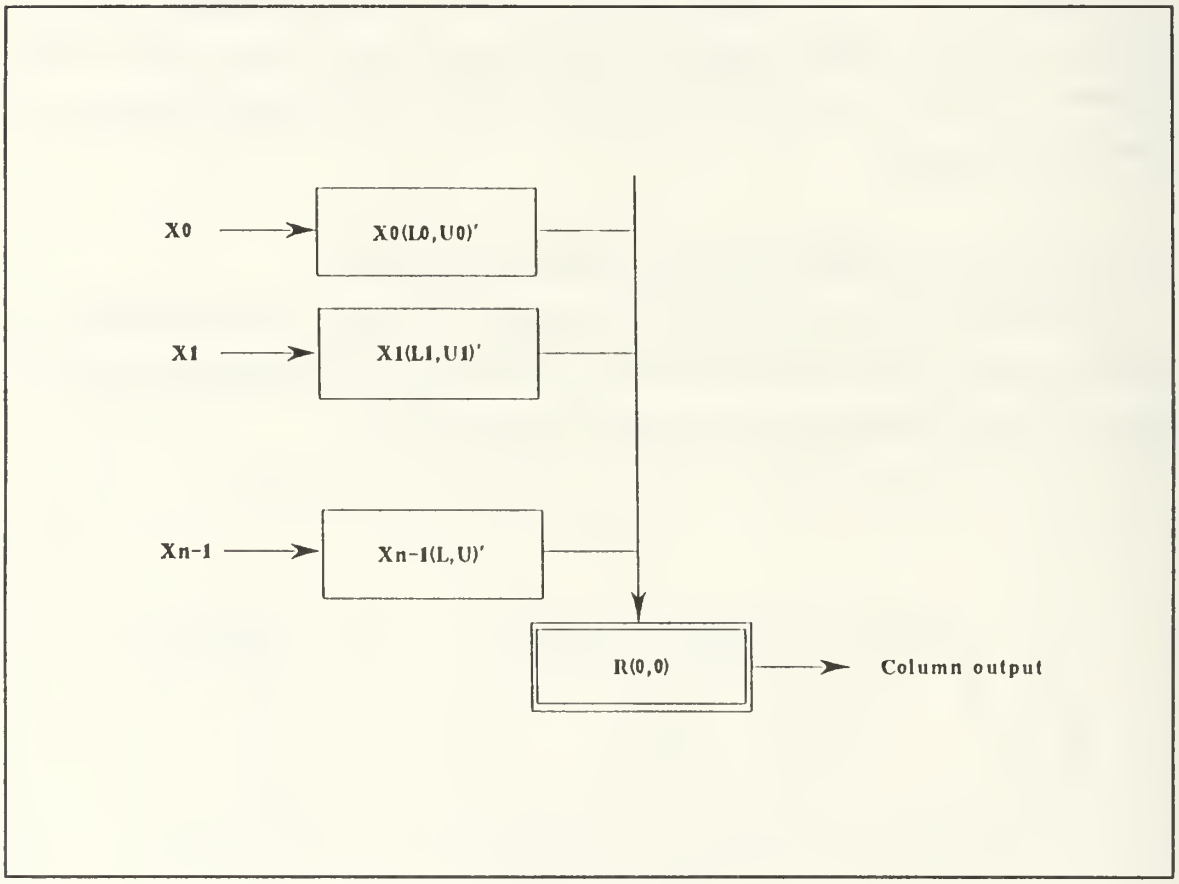


Fig. 5.2 A column cell description.

B. PROGRAM DEVELOPMENT FOR CIRCUIT LAYOUT GENERATION

The circuit layout is generated by a C program named by mvpla (multiple-valued PLA) (Appendix E). The input data file contains the function parameters such as number of inputs, number of outputs and literal function descriptions [Appendix F]. The input data file format is described in Fig. 5.3. The output file contains circuit layout descriptions in MAGIC format (Appendix H) [Ref. 10].

```
4   2   number of inputs,  number of outputs

1       coefficient of 1st product term of 1st function (output)
2   2   lower value, upper value of literal
1   2               .
2   3               .
0   3               .

3       coefficient of 2nd product term
0   3   lower value, upper value of literal
1   1               .
2   2               .
2   3               .
99      end of a function value (99)

2       coefficient of 1st product term of 2nd function (output)
1   3   lower value, upper value of literal
2   2               .
0   3               .
```

Fig. 5.3 Input data file format for mvpla

In Fig. 5.3, the numbers in first 2 column are example function description as 4-valued 4-input 2-output MV-PLA. The first row must include number of variables (inputs) and number of functions (outputs), and each product term has a coefficient and literals. Each function should be terminated by end of function value (99). All values except for the first row and end of function value should be integers in range 0 through 3 for 4-valued function. The number of rows for literal values should be equal to number of inputs, and the number of end of function values (99) should be equal to number of functions (outputs). If the input data file includes invalid values, mvpla will give you error message for the first invalid value and terminate generation immediately.

mvpla uses SCMOS (Scalable CMOS) technology and abides by the MOSSIS design rules so that the circuit layout passes the design rule checker (drc command in MAGIC program). The program does not call library cells or circuit description files. The primary cells reside inside the program, and the leaf cells are created during execution of program.

The primary SYMBOL cells are as follows:

inreplicator : leaf cell for the input replicator,

gen_mvplacell() : leaf cell for the literal function generator,

columnleafcell : leaf cell for a product term or a column,

columnoutput() : leaf cell for the column output generator,

onefunction : leaf cell for a function or a output,

xlabel : label of input,

flabel : label of output,

mvpla : complete circuit layout to be saved in MAGIC format.

Each leaf cell is not built by one step. As shown in program mvpla.c, those cells are created by adding parts as required.

C. SIMULATION OF THE MULTIPLE-VALUED PLA

The circuit layout for multiple-valued PLA is described in MAGIC format. To simulate the circuit layout with SPICE program, first of all, we have to run the MAGIC program, then extract the circuit description file as the simulation input data file. We can obtain the input data file (Appendix G) for SPICE from the extracted circuit description file. SPICE is not a switching level simulation program; currently the switching level multiple-valued simulation program is not available. There are limitations for the analog simulation in terms of the number of transistors and the convergency problem during bias point calculation. However, the verification can be accomplished within those limitations. It is not necessary for PLA cells to be verified by all kinds of MVL functions.

The procedure for the simulation after layout generation is shown below.

1. run the MAGIC program with the output of mvpla (magic)
2. extract the circuit description file from MAGIC program (magic:ext)
3. process the extracted file to obtain simulation file (ext2sim)
4. create a SPICE data file from simulation file (sim2spice)
5. modify a SPICE data file appropriately
6. run SPICE

The programs for these steps are currently available, which are magic, ext2sim, sim2spice and spice. [Ref. 10]

Three example 4-valued PLA circuit layouts were generated by mvpla,

1. 1-input 1-output function (Appendix F),
2. 4-input 2-output function (Appendix H),
3. 8-input 2-output function (Appendix I).

Example 1 was simulated by SPICE program. Example 3 is for the randomly chosen function to see the regularity of layout in general case. The circuit layout does not include the pad frame for the complete form of LSI chip. Addition pad frame to this circuit layout should be done before sending the circuit description file to the fabrication lab. The results of SPICE simulation is shown in Fig. 5.4.

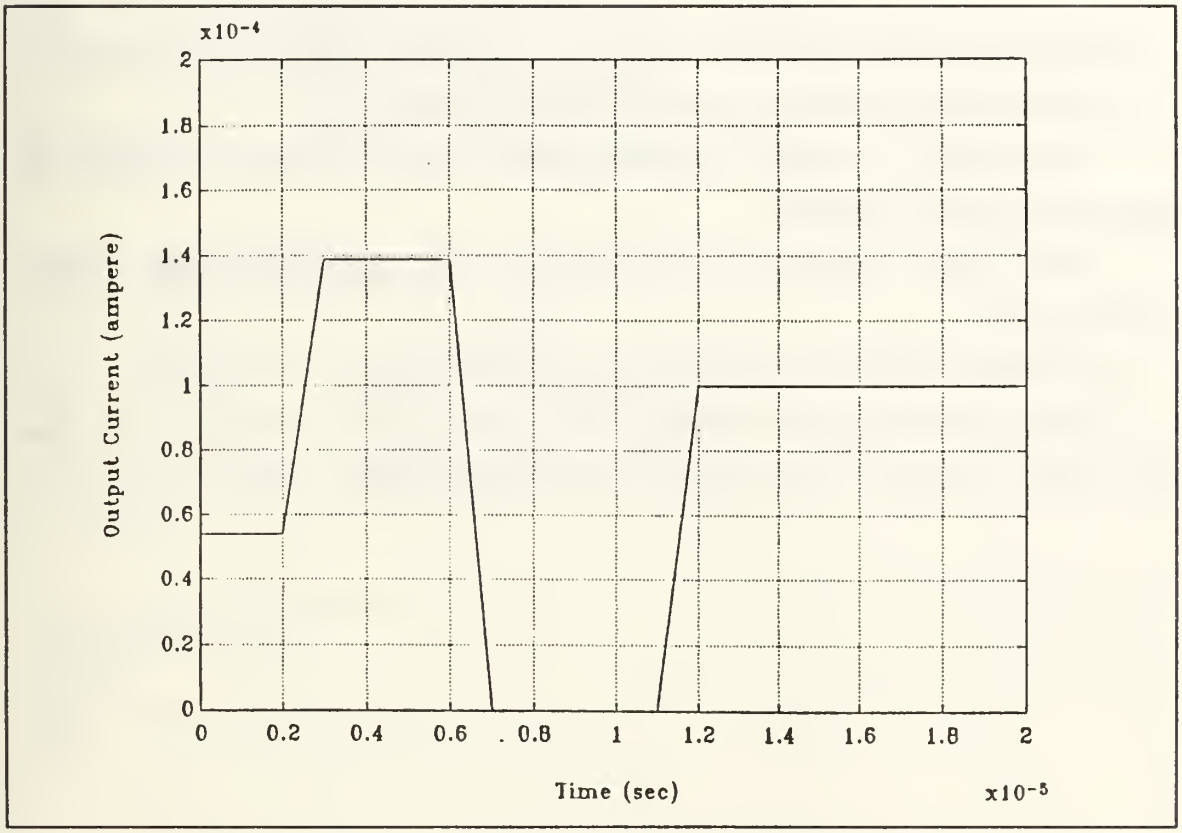


Fig. 5.4 DC transfer characteristic of example 1. circuit.

VI. CONCLUSIONS

The following represent the primary contributions of this thesis.

- The multiple-valued PLA cells are designed and verified by the analog level simulation.

- The mvpla program is developed to generate the PLA circuit layout for 4-valued MVL function.

- The MVL functions are successfully generated by mvpla and some functions which do not violate the limitation of simulation program are simulated by the analog level simulation.

The following recommendations and areas of further investigation are suggested.

- Multiple-valued PLA design with radix higher than 4,

- Development of multiple switching level circuit simulation program for multiple-valued logic circuits,

- Noise margin analysis of multiple-valued logic circuits for each multiple switching level,

- Comparison with binary PLAs in terms of functionality, size and speed,

- Multiple-valued PLA cell design based on voltage mode operations rather than current mode operations show promise of significant reduction in size of layout.

APPENDIX A
PSPICE INPUT DATA FILE FOR INPUT REPLICATOR

```
MV-PLA CELL:Input Replicator
* Lamda based design rule:1.5 Um/Lamda, G = (W/L)
.WIDTH OUT = 80
.OPTIONS NODE LIMPTS=3000 NUMDGT=8 RELTOL=.01
.TRAN 1us 30us

Vdd   1  0 5V
Iin1   0 22 PWL(0us 0uA 30us 300uA)
*
Vin1   22 2
* for equal size of MOSFET
M0    2  2  0  0 CMOSN L=3Um W=4.5Um
M1    3  3  1  1 CMOSP L=3Um W=4.5Um
M2    3  2  0  0 CMOSN L=3Um W=4.5Um
M3    4  3  1  1 CMOSP L=3Um W=4.5Um
MLOAD 5  5  0  0 CMOSN L=3Um W=10.5Um
Vo1 4  5
*
* adjust size of MOSFET
Iin2   0 220 PWL(0us 0uA 30us 300uA)
Vin2   220 20
M00    20  20  0  0   CMOSN L=3Um W=4.5Um
M10    30  30  1  1   CMOSP L=3Um W=9Um
M20    30  20  0  0   CMOSN L=3Um W=4.5Um
M30    40  30  1  1   CMOSP L=3Um W=9Um
MLOAD0 50  50  0  0   CMOSN L=3Um W=10.5Um
Vo2  40 50
*
* call library for model card and subckt
.LIB MVPCELL.LIB
.PROBE
.PRINT TRAN I(Vdum1) I(Vdum2)
.END
```

APPENDIX B
PSPICE INPUT DATA FILE(STEP-UP FUNCTION GENERATOR)

```
MV-PLA Cell:STEP-UP FUNCTION
* Lamda based design rule:1.5 Um/Lamda, G = (W/L)
.WIDTH OUT = 80
.OPTIONS NODE LIMPTS=3000 NUMDGT=8 RELTOL=.01
.TRAN 1us 20us
Iin   0 42 PWL(0us 0uA 20us 200uA)
*
* dummy source to measure current
Vi  42  20
*
Xin  20 9 MVPIN
*
X1  9 11 1  MVP2
M1  11 11  0  0 CMOSN L=9Um W=4.5Um
ML1 100 100 0 0 CMOSN L=3Um W=4.5Um
V1  1 100
*
X2  9 21 2  MVP2
M2  21 21  0  0 CMOSN L=3Um W=4.5Um
ML2 200 200 0 0 CMOSN L=3Um W=4.5Um
V2  2 200
*
X3  9 31 3  MVP2
M3  31 31  0  0 CMOSN L=3Um W=7.5Um
ML3 300 300 0 0 CMOSN L=3Um W=4.5Um
V3  3 300
*
* call library for model card and subckt
.LIB MVPCELL.LIB
.PROBE
.PRINT TRAN I(Vi) I(V1) I(V2) I(V3)
.END
```

APPENDIX C
PSPICE INPUT DATA FILE(STEP-DOWN FUNCTION GENERATOR)

```
MV-PLA Cell:STEP-DONW FUNCTION
* Lamda based design rule:1.5 Um/Lamda, G = (W/L)
.WIDTH OUT = 80
.OPTIONS NODE LIMPTS=3000 NUMDGT=8 RELTOL=.01
.TRAN 1us 20us
Iin   0 42 PWL(0us 0uA 20us 200uA)
*
* dummy source to measure current
Vps 10 0 5
Vi  42  20
*
Xin  20 9 MVPIN
*
X1  9 11 1  MVP2
M1  11 11  0  0 CMOSN L=9Um W=4.5Um
ML1 10 1 100 10 CMOSP L=3Um W=4.5Um
V1  100 0
*
X2  9 21 2  MVP2
M2  21 21  0  0 CMOSN L=3Um W=4.5Um
ML2 10 2 200 10 CMOSP L=3Um W=4.5Um
V2  200 0
*
X3  9 31 3  MVP2
M3  31 31  0  0 CMOSN L=3Um W=7.5Um
ML3 10 3 300 10 CMOSP L=3Um W=4.5Um
V3  300 0
*
* call library for model card and subckt
.LIB MVPCELL.LIB
.PROBE
.PRINT TRAN I(Vi) I(V1) I(V2) I(V3)
.END
```

APPENDIX D
PSPICE INPUT DATA FILE(COLUMN OUTPUT GENERATOR)

```
MV-PLA Cell:COLUMN OUT GENERATOR
* Lamda based design rule:1.5 Um/Lamda, G = (W/L)
.WIDTH OUT = 80
.OPTIONS NODE LIMPTS=3000 NUMDGT=8 RELTOL=.01
.TRAN 1us 20us
Iin    0 42 PWL(0us 0uA 20us 200uA)
*
* dummy source to measure current
Vps 10 0 5
Vi  42  20
*
Xin  20 9 MVPIN
*
X1   9 11 1  MVP2
M1   11 11  0   0 CMOSN L=9Um W=4.5Um
*
ML10 100 1 10 10 CMOSP L=10.5Um W=4.5Um
ML11 100 100 0 0 CMOSN L=3Um W=4.5Um
V1   100 0
*
ML2 200 1 10 10 CMOSP L=6Um W=4.5Um
ML22 200 200 0 0 CMOSN L=3Um W=4.5Um
V2   200 0
*
ML3 300 1 10 10 CMOSP L=4.5Um W=4.5Um
ML33 300 300 0 0 CMOSN L=3Um W=4.5Um
V3   300 0
*
* call library for model card and subckt
.LIB MVPCELL.LIB
.PROBE
.PRINT TRAN I(Vi) I(V1) I(V2) I(V3)
.END
```

```
*****************************************************
*      Subcircuit definitions
*      for  CMOS-MV-PLA Cells
*                   August 25 1988
*****************************************************
.SUBCKT MVPIN   2   3
*              In Out
Vdd 1 0 DC 5
M01 2 2 0 0 CMOSN L=3Um W=9Um
M02 3 2 0 0 CMOSN L=3Um W=9Um
M03 3 3 1 1 CMOSP L=3Um W=10.5Um
.ENDS
*
.SUBCKT MVP1     3          11        12
*            Input_gate   NMOS_D&G   Out
* Single invertor
Vdd 1 0 DC 5
M11   11   3 1 1 CMOSP L=3Um W=10.5Um
M13   12 11 1 1 CMOSP L=3Um W=4.5Um
M14   12 11 0 0 CMOSN L=3Um W=4.5Um
.ENDS
*
.SUBCKT MVP2     3          11        13
*            Input_gate NMOS_D&G   Out
* Double invertor
Vdd 1 0 DC 5
M11   11   3 1 1 CMOSP L=3Um W=10.5Um
M13   12 11 1 1 CMOSP L=3Um W=4.5Um
M14   12 11 0 0 CMOSN L=3Um W=4.5Um
M15   13 12 1 1 CMOSP L=3Um W=4.5Um
M16   13 12 0 0 CMOSN L=3Um W=4.5Um
.ENDS
*
*************************************************************
*    Pspice MODEL parameters for NMOS and PMOS transistor
*************************************************************
.MODEL CMOSN NMOS(LEVEL = 3
+TPG         = 1
+VTO         = 0.62
+KP          = 6.93E-05
+GAMMA       = 0.73
+PHI         = 0.600
+TOX         = 25NM
+NSUB        = 2.24E17
+NFS         = 1E10
+VMAX        = 2.09E05
+ETA         = 0.100
```

```
+DELTA      = 0.211
+THETA      = 3.47E-02
+KAPPA      = 8.83
+CGSO       = 1.2E-10
+CGDO       = 1.2E-10
+CGBO       = 3.4E-10
+RSH        = 24
+JS         = 1.5E-5
+XJ         = 3.50E-07
+LD         = 2.27E-07
+CJ         = 3.36E-4
+MJ         = 0.97
+CJSW       = 1.34E-10
+MJSW       = 0.65
+PB         = 0.94)
*****************************************************************
.MODEL CMOSP PMOS(LEVEL = 3
+TPG        = -1
+VTO        = -0.84
+KP         = 2.15E-05
+GAMMA      = 0.57
+PHI        = 0.700
+TOX        = 25NM
+NSUB       = 3.07E16
+NFS        = 1E10
+VMAX       = 2.14E05
+ETA        = 0.208
+DELTA      = 0.121
+THETA      = 5.97E-02
+KAPPA      = 8.00
+CGSO       = 1.7E-10
+CGDO       = 1.7E-10
+CGBO       = 3.4E-10
+RSH        = 67
+JS         = 1E-6
+XJ         = 1.69E-07
+LD         = 3.30E-07
+CJ         = 7.27E-4
+MJ         = 0.41
+CJSW       = 2.90E-10
+MJSW       = 0.37
+PB         = 0.91)
```

APPENDIX E
PROGRAM FOR 4-VALUED MV-PLA CIRCUIT GENERATION

```
/************************************************************
        This is the header file for mvpla.c

        written by Ko, Y. H.
                ECE Department. NPS
        Nov. 22 1988
************************************************************/

#include "/tools/uw/include/cfl.h"
#include "stdio.h"
#include "strings.h"
#include "ctype.h"

#define TRUE  1
#define FALSE 0

#define MIN   0
#define MAX   3

#define FOR_A_FUN 99
#define MAXINPUTS 32
#define MAXOUTPUTS 32
#define MAXPRODUCTS 99

/* min and max size(in lamda) of transistor */
#define WMIN   3
#define LMIN   2
#define WMAX   10
#define LMAX   6

/* define name of layers */
#define M1 "metal1"
#define M2 "metal2"
#define ND "ndiffusion"
#define PD "pdiffusion"
#define POLY "polysilicon"
#define NDC "ndcontact"
#define PDC "pdcontact"
#define NSC "nsubstratencontact"
#define PSC "psubstratepcontact"
```

```
#define M2C "m2contact"
#define PMC "polycontact"

/* file definitions */
FILE *infile,*outfile;

/* file names */
char *infilename, *outfilename;

/* leaf cells */
SYMBOL *nmos,*pmos,*cmos,*inv1,*inv2,*stepdn,*stepup;
SYMBOL *cmirror,*cinput,*nout,*pout;

/* layer variables */
SYMBOL *m1,*m2,*poly,*nd,*pd,*ndc,*pdc,*nsc,*psc,*pc,*m2c;

/* symbol for literal function generator */
SYMBOL *mvplacell[4][4];

/* name of literal function generator cells = magic filename for leaf cells */
char *mvcell[4][4] = {
                        {"mvcell00","mvcell01","mvcell02","mvcell03"},
                        {"mvcell10","mvcell11","mvcell12","mvcell13"},
                        {"mvcell20","mvcell21","mvcell22","mvcell23"},
                        {"mvcell30","mvcell31","mvcell32","mvcell33"}
                      };

/* labels */
SYMBOL *vdd,*gnd,*inlabel,*outlabel;

/* functions with return value type SYMBOL */
SYMBOL  *gen_mvplacell(),
        *downout(),
        *upout(),
        *inputcell(),
        *columnoutput();

/* struct of input variable */
typedef struct {
                    short l,   /* x(l,u) */
                          u;
                } INPUTS;

/* struct of a product term */
typedef struct {
                    short coeff;
                    INPUTS x[MAXINPUTS];
```

```
                } PRODUCTTERM;

/* struct of a function */
typedef struct {
                  PRODUCTTERM pterm[MAXPRODUCTS];
                } ONEFUNCTION;

/* function array */
ONEFUNCTION fun[MAXOUTPUTS];

/* number of variables */
short nvar;

/* number of functions */
short nfun;
```

```
/*****************************************************************

   This is the program MVPLA.C to generate the multiple valued logic
   PLA layout in magic format with scmos technology.

                                   Written by Ko, Y. H.
                                   ECE Department  NPS

                                   November 22  1988

*****************************************************************/

#include "mvpla.h"

main(argc,argv)
   int argc;
   char *argv[];
{

   /* boolean variables for in out file on command line */
   short infileexist = FALSE;

   /* variable for logic level */
   int level;

   printf("\n\n\nmvpla <4-valued> - Version 1.01 - Last updated 11/17/88\n\n");

   /* check the command line input and process it if exist */
   if (argc == 2)
   {
      /* opne a file */
      infilename = argv[1];

      /* assign outfile name same as input filename */
      outfilename = argv[1];

   }
   else if (argc >= 3)
   {
      /* open file */
      infilename = argv[1];

      /* assign user defined outfile name */
      outfilename = argv[2];

   }
   else
```

```
    {
      printf("\n\nUsage: mvpla infile [outfile]\n\n");
      exit(1);
    }

    printf("\nInput  file : '%s'",infilename);
    printf("\nOutput file : '%s.mag'",outfilename);

    /* set output format */
    cflsetc("format","magic");

    /* define technology */
    cflstart("scmos");

    /* set lamda size equal to the magic unit */
    cflsetv("grain",1);

    /* generate layout complete */
    generatelayout(infilename,outfilename);

    /* exit cfl */
    cflstop();

    puts("\nDone !\n");

}/* end of main */


/*****************************************************************

    This routine generate complete MV-PLA layout.

*****************************************************************/

generatelayout(infl,outfl)
    char *infl;
    char *outfl;

{
    /* loop variables */
    short i,j=0,k,m;

    /* input label name */
    char *xlabel = "x00",*flabel = "f00";

    /* variables for lower and upper value of input and coefficient */
    short lower,upper,coe;
```

```
/* leaf cells */
SYMBOL *onefunction,*tmpcell,*inreplicator,*columnleafcell,*cout,*mvpla;

/* open input file */
infile = fopen(infl,"r");

/* read the input data from infile, if invalid data, then errorexit */
getdatafile(infile);

/* generate complete MV_PLA layout */
puts("\n\nNow, generating CMOS-MV-PLA layout in magic format.....");

/* draw the basic cells first with the given level */
createcells();

for (i=0;i<4;i++)
{
   for (j=i;j<4;j++)
   {
      ps(mvcell[i][j],gen_mvplacell(i,j));
      mvplacell[i][j] = gs(mvcell[i][j]);
   }
}

inlabel = mlabel(xlabel,0,0,"top",M1);
cmirror = my(cxdx(inlabel,cmirror,-4));

/* input part of complete layout */
inreplicator = cmirror;

/* input replicators */
for (m = 1;m < nvar;m++)
{
   /* change name of input label, x00..x99 */
   xlabel[1] = '0' + m / 10;
   xlabel[2] = '0' + m % 10;

   inlabel = mlabel(xlabel,0,0,"top",M1);
   cmirror = my(cxdx(inlabel,cmirror,-4));

   /* stack down input cells */
   inreplicator = rrdy(cmirror,inreplicator,-4);

}/*for m*/

/* add Vdd for columnoutput generator */
```

```
inreplicator = rrdy(box(M2,32,4),inreplicator,12);

/* put Vdd line with label */
inreplicator = ttdx(cydy(box(M2,8,nvar*42+24),vdd,-4),inreplicator,-18);

/* dummy lablel */
mvpla = mlabel("m",0,0,"top",M2);

for (i = 0;i < nfun;i++)
{
   printf("\n\nf%ld = ",i);

   /* initialize num of product term zero */
   j = 0;

   onefunction = mlabel("f",0,0,"top",M2);

   while (fun[i].pterm[j].coeff != FOR_A_FUN)
   {
      /* build column leaf cell */
      lower = fun[i].pterm[j].x[0].l;
      upper = fun[i].pterm[j].x[0].u;
      columnleafcell = mvplacell[lower][upper];
      columnleafcell = my(columnleafcell);

      /* echo given function to confirm */
      printf("%hdx0(%hd,%hd)",fun[i].pterm[j].coeff,lower,upper);

      for (k = 1;k < nvar;k++)
      {
         lower = fun[i].pterm[j].x[k].l;
         upper = fun[i].pterm[j].x[k].u;
         tmpcell = mvplacell[lower][upper];

         printf("x%ld(%hd,%hd)",k,lower,upper);

         /* upsidedown odd cell */
         if ((1+k)%2) tmpcell = my(tmpcell);

         /* stack down cells for input x[0] on top */
         columnleafcell = rrdy(tmpcell,columnleafcell,-4);
      }/*for k*/

      /* put column output generator */
      cout = columnoutput(fun[i].pterm[j].coeff);
      columnleafcell = rr(cout,columnleafcell);
```

```
        /* build a function by putting product terms */
        onefunction = tt(onefunction,columnleafcell);

        j++;

        /* a column leaf cell generated */
        if (fun[i].pterm[j].coeff != FOR_A_FUN)
        {
           printf("\n    + ");
        }

    }/*while*/

    /* change name of input label, x00..x99 */
    flabel[1] = '0' + i / 10;
    flabel[2] = '0' + i % 10;

    /* put a wired-sum part */
    outlabel = mlabel(flabel,0,0,"top",M1);
    onefunction = rrdx(cc(outlabel,box(M1,(j-1)*98+6,8)),onefunction,14);

    /* put functions together */
    mvpla = tt(mvpla,onefunction);

  }/*for i*/

/* put input replicators at left side of layout */
mvpla = tt(inreplicator,mvpla);

/* put GND base line at the right side of layout */
tmpcell = cydy(box(M2,8,4),box(M2,8,8),72);

if (nvar == 1)
     tmpcell = cy(box(M2,8,8),tmpcell);
else if (nvar % 2) /* odd number of inputs */
     tmpcell = cy(box(M2,8,8),ny(tmpcell,nvar/2));
else
     tmpcell = cy(box(M2,8,4),ny(tmpcell,nvar/2));

tmpcell = tt(tmpcell,cydy(box(M2,8,nvar*42+66),gnd,-4));

/* complete layout */
mvpla = bbdy(mvpla,box(M2,8,4),16);
mvpla = ttdx(mvpla,tmpcell,-8);

/* save leaf cells in 'mag' dir */
ps(outfl,mvpla);
```

```
    printf("\n\nOutput to %s.mag...\n",outfl);

}/* generatelayout */

/*****************************************************************

    This routine generate complete MV-PLA Cell.
    lower, upper : logic value(i.e., 0,1,2,3)

*****************************************************************/

SYMBOL *gen_mvplacell(l,u)
    short l,u;
{

    short len,wid;

    /* declare local symbol vars */
    SYMBOL *outcell,*columnout;

    createcells();

    columnout = box(M1,4,46);

    /* configure up and down step function with the basic cell */

    if ((l > MIN) && (u == MAX))
    {
        /* step down cell */
        getdevsize(&wid,&len,l-0.5);
        outcell = tt(bb(inputcell(wid,len),inv2),downout(54));
        outcell = cc(outcell,columnout);
    }
    else if ((l == MIN) && (u < MAX))
    {
        /* step up cell */
        getdevsize(&wid,&len,u+0.5);
        outcell = ttdy(bb(inputcell(wid,len),inv2),upout(54),-4);
        outcell = cc(outcell,columnout);
    }
    else if ((l <= MIN) && (u >= MAX))
    {
        outcell = cydy(box(M2,92,4),box(M2,92,4),15);
        outcell = cydy(outcell,box(M2,92,4),11);
        outcell = cc(outcell,columnout);
    }
```

```
    else
    {
        /* up and down together */
        getdevsize(&wid,&len,l-0.5);
        stepdn = tt(bb(inputcell(wid,len),inv2),downout(8));

        getdevsize(&wid,&len,u+0.5);
        stepup = mx(ttdy(bb(inputcell(wid,len),inv2),upout(8),-4));

        outcell = bb(stepdn,stepup);
        outcell = cc(outcell,columnout);
    }

    /* put common gate input */
    columnout = rr(rr(rr(m2c,cc(box(M1,4,1),box(M2,4,1))),
                        cc(box(M2,4,4),pc)),box(POLY,4,10));
    outcell = ttdy(outcell,columnout,-9);
    outcell = rrdy(outcell,box(M2,4,4),-8);
    outcell = rrdy(box(M2,4,4),outcell,-8);
    outcell = cydy(cydy(box(M2,96,4),outcell,-4),box(M2,96,4),-4);

    /* return complete cell */
    return(outcell);

}/* gen_mvplacell */


/*****************************************************************

    IN : logic level/threshold
    OUT: geometry of device in lambda(width/length)

*****************************************************************/
getdevsize(wid,len,value)
    short *wid,*len;
    float value;
{
    if ((value == 0.5) || (value == 1))
    {
        *wid = 3;
        *len = 3 / value;
    }
    else
    {
        *wid = 2 * value;
        *len = 2;
    }
```

```
}/*getdevsize*/

/*****************************************************************

   This routine read the data from infile.

*****************************************************************/

getdatafile(fn)
        FILE *fn;
{
   int tmpcoeff,lower,upper;

   /* loop variables and line counter of input data file */
   int i,j,k,nline=1;

   /* read number of inputs and outputs */
   if(EOF == fscanf(fn,"%hd %hd",&nvar,&nfun))
   {
      printf("\n\n***ERROR*** '%s', Line %d : ",infilename,nline);
      printf("Unexpected end of file, Need more values.\n\n");
      exit();
   }

   printf("\n\n\nCheck: %hd input(s) %hd output(s) MV-PLA",nvar,nfun);

   if ((nvar <= 0) || (nfun <= 0))
   {
      printf("\n\n***ERROR*** '%s', Line %d : ",infilename,nline);
      printf("Invaid number of variables or functions.\n\n");
      exit();
   }
   else if ((nvar > MAXINPUTS) || (nfun > MAXOUTPUTS))
   {
      printf("\n\n***WARNING*** '%s', Line %d : ",infilename,nline);
      printf("Too many inputs or outputs.\n\n");
      exit();
   }


   /* read functions */
   for (i = 0;i < nfun;i++)
   {
      /* function name = output label */
      printf("\n\nf%d = ",i);

      /* initialize num of product term as zero */
```

```
j = 0;

/* read first coefficient of product term as a sentinal */
if(EOF == fscanf(fn,"%hd",&tmpcoeff))
{
   printf("\n\n***ERROR*** '%s', Line %d : ",infilename,nline);
   printf("Unexpected end of file, Need more values.\n\n");
   exit();
}

fun[i].pterm[j].coeff = tmpcoeff;

/* increment line counter */
nline++;

/* echo print input values to check */
printf("%hd",tmpcoeff);

if ((tmpcoeff <= 0) || (tmpcoeff > MAX))
{
   puts("\v\b^");
   printf("\n***ERROR*** '%s', Line %d : ",infilename,nline);
   printf("Invalid coefficient.\n\n");
   exit();
}

while (tmpcoeff != FOR_A_FUN)
{
   for (k = 0;k < nvar;k++)
   {
      /* read lower and upper of a variable */
      if(EOF == fscanf(fn,"%d %d",&lower,&upper))
      {
         printf("\n\n***ERROR*** '%s', Line %d : ",infilename,nline);
         printf("Unexpected end of file, Need more values.\n\n");
         exit();
      }

      /* increment line counter */
      nline++;

      /* echo print input values to check */
      printf("x%ld(%hd,%d)",k,lower,upper);

      /* check data validity and exit with error message if error */
      if ((lower > upper) || (lower < MIN) || (upper > MAX))
      {
```

```
      puts("\v\b\b\b^");
      printf("\n***ERROR*** '%s', Line %d : ",infilename,nline);
      printf("Invalid values.\n\n");
      exit();
   }

   /* update global var's fields */
   fun[i].pterm[j].x[k].l = lower;
   fun[i].pterm[j].x[k].u = upper;

}/*for k*/

/* count number of product terms in a function */
j++;

/* read next coefficient of product term and inputs */
if(EOF == fscanf(fn,"%d",&tmpcoeff))
{
   printf("\n\n***ERROR*** '%s', Line %d : ",infilename,nline);
   printf("Unexpected end of file, Need more values.\n\n");
   exit();
}

/* increment line counter */
nline++;

fun[i].pterm[j].coeff = tmpcoeff;

if (tmpcoeff != FOR_A_FUN)
{
   /* echo print input values to check */
   printf("\n    + %hd",tmpcoeff);

   if ((tmpcoeff <= 0) || (tmpcoeff > MAX))
   {
      puts("\v\b^");
      printf("\n***ERROR*** '%s', Line %d : ",infilename,nline);
      printf("Invalid coefficient.\n\n");
      exit();
   }

   if (j > MAXPRODUCTS)
   {
      printf("\n\n***WARNING*** '%s', Line %d : ",infilename,nline);
      printf("Too many product terms<MAX 99>.\n\n");
      exit();
   }
```

```
            }/* if */

        }/*while*/

    }/*for*/

}/* end of getdatafile */

/*****************************************************************

    This routine generate the basic cell for the step up function
    with the given logic level.

    cells : inv1, inv2, cmirror.

*****************************************************************/

createcells()
{
    /* local leaf cells */
    SYMBOL *m1wire,*polywire;

    /* symbols*/
    nd = box(ND,3,4);
    pd = box(PD,3,4);
    ndc = box(NDC,4,4);
    pdc = box(PDC,4,4);
    nsc = box(NSC,4,4);
    psc = box(PSC,4,4);
    m2c = box(M2C,4,4);
    pc  = box(PMC,4,4);
    m1wire = box(M1,4,14);

    /* labels */
    vdd = mlabel("Vdd!",0,0,"top",M2);
    gnd = mlabel("GND!",0,0,"top",M2);

    /* put things together */
    polywire = ll(ll(box(POLY,8,2),box(POLY,2,24)),box(POLY,8,2));
    nmos = cy(cydx(ndc,nd,-1),ndc);
    pmos = cy(cydx(pdc,pd,-1),cc(pdc,box(M1,10,4)));
    inv1 = cy(cy(cy(cy(psc,nmos),m1wire),pmos),nsc);

    /* put poly contact on output */
    inv1 = ccdxy(ccdx(inv1,polywire,-1),pc,3,2);
```

```
    /* series of inverter and add m2 contact and GND line */
    inv2 = tt(cydy(box(M1,10,4),inv1,-8),lldy(box(M1,7,4),inv1,-8));
    inv2 = cydy(cc(m2c,box(M2,20,4)),inv2,-8);
    inv2 = ccdy(inv2,box(M2,20,4),2);
    inv2 = cydy(inv2,cc(m2c,box(M2,20,4)),-8); /* complete */

    /* create cmirror cell */
    /* prepare elements */
    nd = box(ND,6,4);
    pd = box(PD,7,4);
    ndc = box(NDC,6,4);
    pdc = box(PDC,7,4);
    nsc = box(NSC,7,4);
    psc = box(PSC,6,4);

    /* build input current mirror cell */
    cmirror = ll(ll(ll(ll(psc,ndc),nd),ndc),box(M1,6,11));
    cmirror = bb(bbdy(cmirror,cc(box(M1,6,4),m2c),4),cmirror);
    pmos = rr(rr(rr(rr(box(M1,6,3),pdc),pd),pdc),nsc);
    cmirror = rr(cmirror,pmos);

    /* input gates poly */
    polywire = cydx(box(POLY,22,2),rr(box(POLY,2,12),pc),-2);
    cmirror = cydy(polywire,cmirror,-27);

    /* output gates poly */
    polywire = cc(bb(bb(m2c,box(M1,1,4)),pc),box(M2,9,4));
    cmirror = cxdxy(cmirror,polywire,-8,2);
    cmirror = rrdy(cmirror,box(M1,4,4),-8);
    cmirror = rrdy(cmirror,rr(box(POLY,2,8),box(POLY,12,2)),-19);

    /* put GND and input line */
    cmirror = rrdy(box(M2,26,8),cmirror,-8);
    cmirror = cxdx(box(M1,30,8),cmirror,-6);
    cmirror = rrdy(cmirror,box(M2,32,8),-8);

}/* end of createcells */
```

```
/*****************************************************************

   This routine generate the basic cell for the output part.

   cell : pout,nout

*****************************************************************/

SYMBOL *downout(w)
   short w;
{

   SYMBOL *polywire;

   /* symbols*/
   pd = box(PD,3,4);
   pdc = box(PDC,4,4);
   nsc = box(NSC,4,4);

   pmos = cy(cydx(pdc,pd,-1),cc(pdc,box(M1,10,4)));
   pmos = rr(nsc,pmos);

   /* output part of cell */
   pmos = tt(box(M1,3,4),ll(ll(pdc,pd),pdc));

   pout = bb(rr(pmos,nsc),box(M1,4,4));
   pout = lldy(pout,ll(box(POLY,2,12),box(POLY,8,2)),-23);

   pout = lldy(pout,box(M2,w+4,4),-8);
   pout = lldy(pout,box(M1,3,4),-8);
   pout = lldy(pout,box(M2,w+4,4),-23);
   pout = lldy(pout,box(M2,w+4,4),-42);

   return (pout);

}/* downout */

SYMBOL *upout(w)
   short w;
{
   SYMBOL *polywire;

   /* symbols*/
   nd = box(ND,3,4);
   ndc = box(NDC,4,4);

   nmos = cy(cydx(ndc,nd,-1),ndc);
```

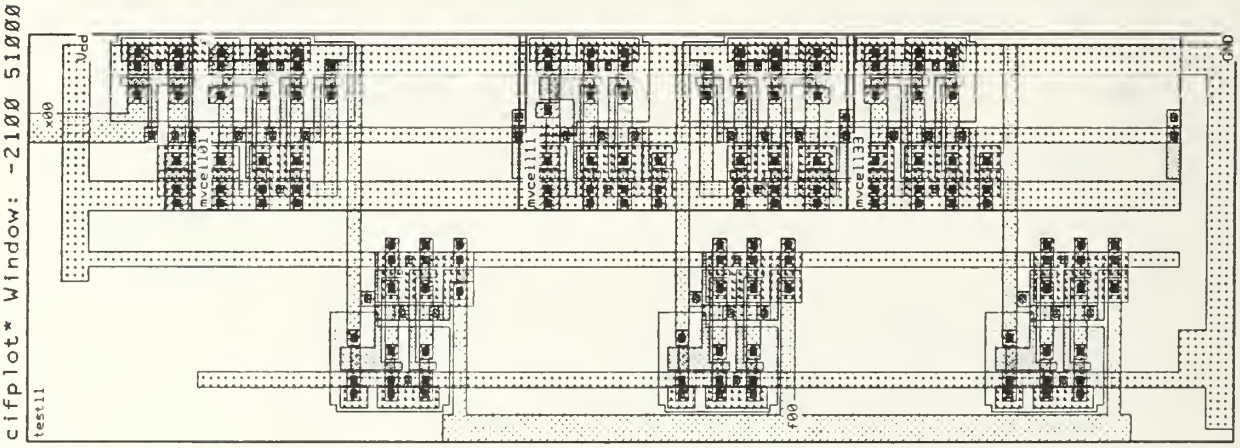
```
    /* nmos output */
    nmos = bb(nmos,box(M1,4,4));

    nout = tt(box(M1,3,4),box(M1,4,26));
    nout = lldy(bb(box(POLY,2,18),box(POLY,6,2)),lldx(nmos,nout,-3),-23);
    nout = lldy(nout,box(M2,w+4,4),-4);
    nout = lldy(nout,box(M2,w+4,4),-19);
    nout = lldy(box(M2,w+4,4),nout,-4);

    return (nout);

}/* upout */
/*****************************************************************

    This routine generate the basic cell for the input part.

    cell : cinput

*****************************************************************/

SYMBOL *inputcell(w,l)
    short w,l;
{
    /* cell width and contact width */
    short wmax,wc,wmet;

    /* decide max width of cell */
    if (w > WMAX-3)
    {
        wmax = wc = w;
        wmet = 7;
    }
    else if (w <= 4)
    {
        wmax = 7;
        wc = wmet = 4;
    }
    else
    {
        wc   = wmet = w;
        wmax = 7;
    }

    /* symbols*/
    nd  = box(ND,w,2+l);
    pd  = box(PD,7,4);
```

```
    ndc = box(NDC,7,4);/*wc=7*/
    pdc = box(PDC,7,4);

    nsc = box(NSC,7,4);
    psc = box(PSC,7,4);/*wc=7*/

    m2c = box(M2C,4,4);
    pc  = box(PMC,4,4);

    pmos = rr(rr(rr(pdc,pd),pdc),nsc);
    nmos = rr(rr(rr(psc,ndc),nd),ndc);

    cinput = rr(rr(nmos,box(M1,wmet,16-l)),pmos);
    cinput = cxdxy(bbdy(cinput,box(M1,4,4),4),pc,-4,2);

    cinput = rrdy(cinput,box(M1,wmax+7,4),-8);
    cinput = cydy(box(M2,wmax+7,4),cinput,-8);

    cinput = lldy(cinput,box(POLY,wmax+5,2),-11);
    cinput = rrdy(box(POLY,wc+6,l),cinput,-9-l);

    cinput = rrdy(cinput,box(M2,wmax+7,4),-8);
    cinput = rrdy(cinput,box(M2,wmax+7,4),-23); /* complete */

    /* return complete input part as specified */
    return (cinput);

}/* inputcell */


/*****************************************************************

    This routine generate the basic cell for the column output part.

    cell : columnoutput

*****************************************************************/
SYMBOL *columnoutput(value)
    short value;
{
    SYMBOL *out,*temp;

    short wa,la;

    float outlevel;
```

```
    /* input part */
    ndc = box(NDC,4,4);
    psc = box(PSC,4,4);
    out = cy(rr(cy(psc,ndc),box(ND,3,8)),ndc);
    out = cxdy(rr(out,box(M1,4,34)),pc,2);
    out = rrdy(box(M1,4,4),out,-8);
    out = rrdy(box(POLY,10,6),out,-15);

    /* add 2 inverter */
    out = bb(out,inv2);
    out = ttdy(out,box(POLY,2,14),-17);

    /* output part */
    /* get size of transistor as voltage-controlled current source */
    outlevel = value;
    getdevsize(&wa,&la,outlevel);

    if (wa <= 4)
    {
       pdc = box(PDC,4,4);
       nsc = box(NSC,4,4);
    }
    else
    {
       pdc = box(PDC,wa,4);
       nsc = box(NSC,wa,4);
    }

    temp = rr(rr(rr(pdc,box(PD,wa,la+2)),tt(box(M1,3,4),pdc)),nsc);
    temp = rr(box(M1,4,36-la),temp);
    temp = rrdxy(temp,box(POLY,wa+4,la),2,-9-la);

    /* connect output part */
    out = bbdxy(out,temp,-2,-4);


    /* add Vdd and GND lines */
    out =cydy(box(M2,96,4),ttdxy(box(M2,96,4),out,-54,12),-12);

    return(out);

}/* columnoutput */
```

APPENDIX F

CIRCUIT LAYOUT GENERATED BY PROGRAM(mvpla) for 1-INPUT 1-OUTPUT 4-VALUED PLA



APPENDIX G

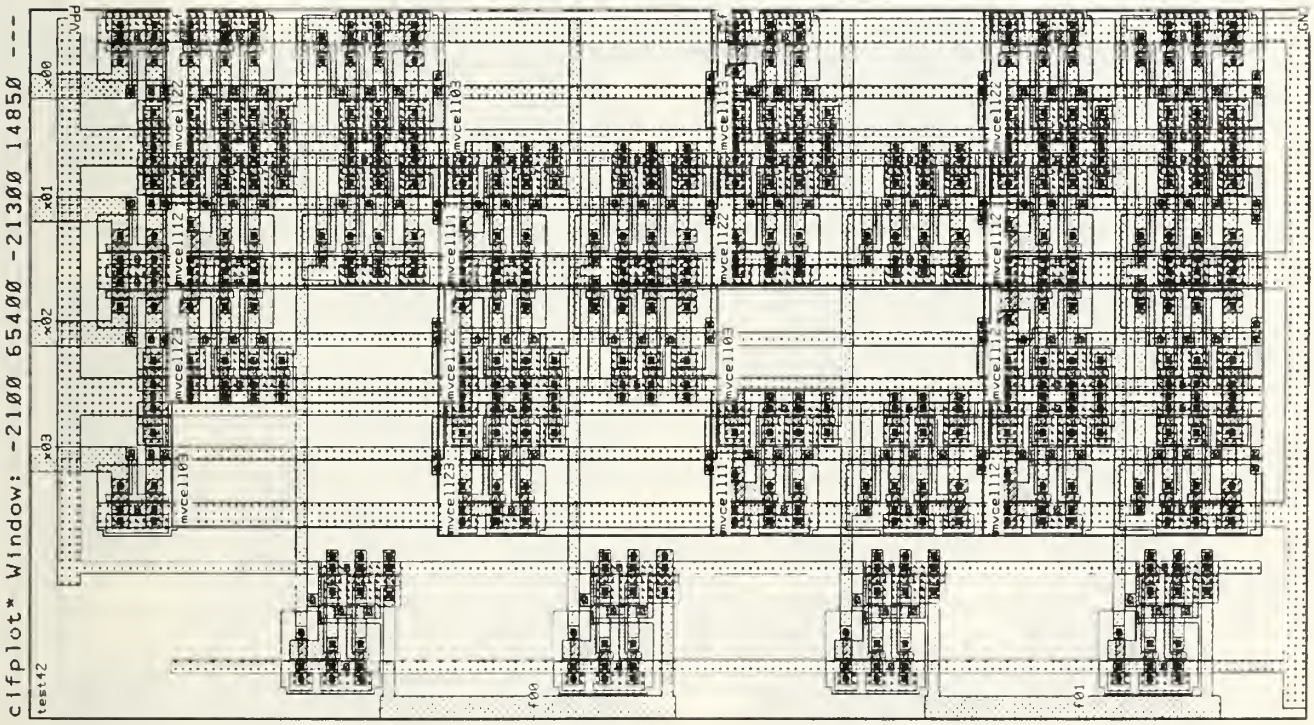
PSPICE INPUT DATA EXTRACTED FROM CIRCUIT LAYOUT

```
*** SPICE DECK created from test11.sim, tech=scmos
M1 6 5 1 4 CMOSP L=3.0U W=10.5U
M2 7 6 1 4 CMOSP L=3.0U W=4.5U
M3 8 7 1 4 CMOSP L=3.0U W=4.5U
M4 9 8 1 4 CMOSP L=3.0U W=4.5U
M5 0 6 6 10 CMOSN L=3.0U W=7.5U
M6 0 6 7 10 CMOSN L=3.0U W=4.5U
M7 0 7 8 10 CMOSN L=3.0U W=4.5U
M8 11 5 1 4 CMOSP L=3.0U W=10.5U
M9 12 11 1 4 CMOSP L=3.0U W=4.5U
M10 13 12 1 4 CMOSP L=3.0U W=4.5U
M11 14 13 1 4 CMOSP L=3.0U W=4.5U
M12 16 15 1 4 CMOSP L=3.0U W=4.5U
M13 15 17 1 4 CMOSP L=3.0U W=4.5U
M14 17 5 1 4 CMOSP L=3.0U W=10.5U
M15 0 11 11 10 CMOSN L=9.0U W=4.5U
M16 0 11 12 10 CMOSN L=3.0U W=4.5U
M17 0 12 13 10 CMOSN L=3.0U W=4.5U
M18 14 16 1 10 CMOSN L=3.0U W=4.5U
M19 0 15 16 10 CMOSN L=3.0U W=4.5U
M20 0 17 15 10 CMOSN L=3.0U W=4.5U
M21 0 17 17 10 CMOSN L=3.0U W=4.5U
M22 18 5 1 4 CMOSP L=3.0U W=10.5U
M23 19 18 1 4 CMOSP L=3.0U W=4.5U
M24 20 19 1 4 CMOSP L=3.0U W=4.5U
M25 0 18 18 10 CMOSN L=3.0U W=4.5U
M26 0 18 19 10 CMOSN L=3.0U W=4.5U
M27 0 19 20 10 CMOSN L=3.0U W=4.5U
M28 21 20 1 10 CMOSN L=3.0U W=4.5U
M29 22 22 0 10 CMOSN L=3.0U W=9.0U
M30 5 22 0 10 CMOSN L=3.0U W=9.0U
M31 1 5 5 4 CMOSP L=3.0U W=10.5U
M32 23 21 1 4 CMOSP L=3.0U W=4.5U
M33 24 23 1 4 CMOSP L=3.0U W=4.5U
M34 0 21 21 10 CMOSN L=9.0U W=4.5U
M35 25 24 1 4 CMOSP L=4.5U W=4.5U
M36 26 14 1 4 CMOSP L=3.0U W=4.5U
M37 27 26 1 4 CMOSP L=3.0U W=4.5U
M38 25 27 1 4 CMOSP L=3.0U W=6.0U
```

```
M39 28 9 1 4 CMOSP L=3.0U W=4.5U
M40 29 28 1 4 CMOSP L=3.0U W=4.5U
M41 25 29 1 4 CMOSP L=3.0U W=6.0U
M42 0 21 23 10 CMOSN L=3.0U W=4.5U
M43 0 23 24 10 CMOSN L=3.0U W=4.5U
M44 0 14 14 10 CMOSN L=9.0U W=4.5U
M45 0 14 26 10 CMOSN L=3.0U W=4.5U
M46 0 26 27 10 CMOSN L=3.0U W=4.5U
M47 0 9 9 10 CMOSN L=9.0U W=4.5U
M48 0 9 28 10 CMOSN L=3.0U W=4.5U
M49 0 28 29 10 CMOSN L=3.0U W=4.5U
C50 25 0 119.0F
C51 5 0 135.0F
C52 1 0 1113.0F
```

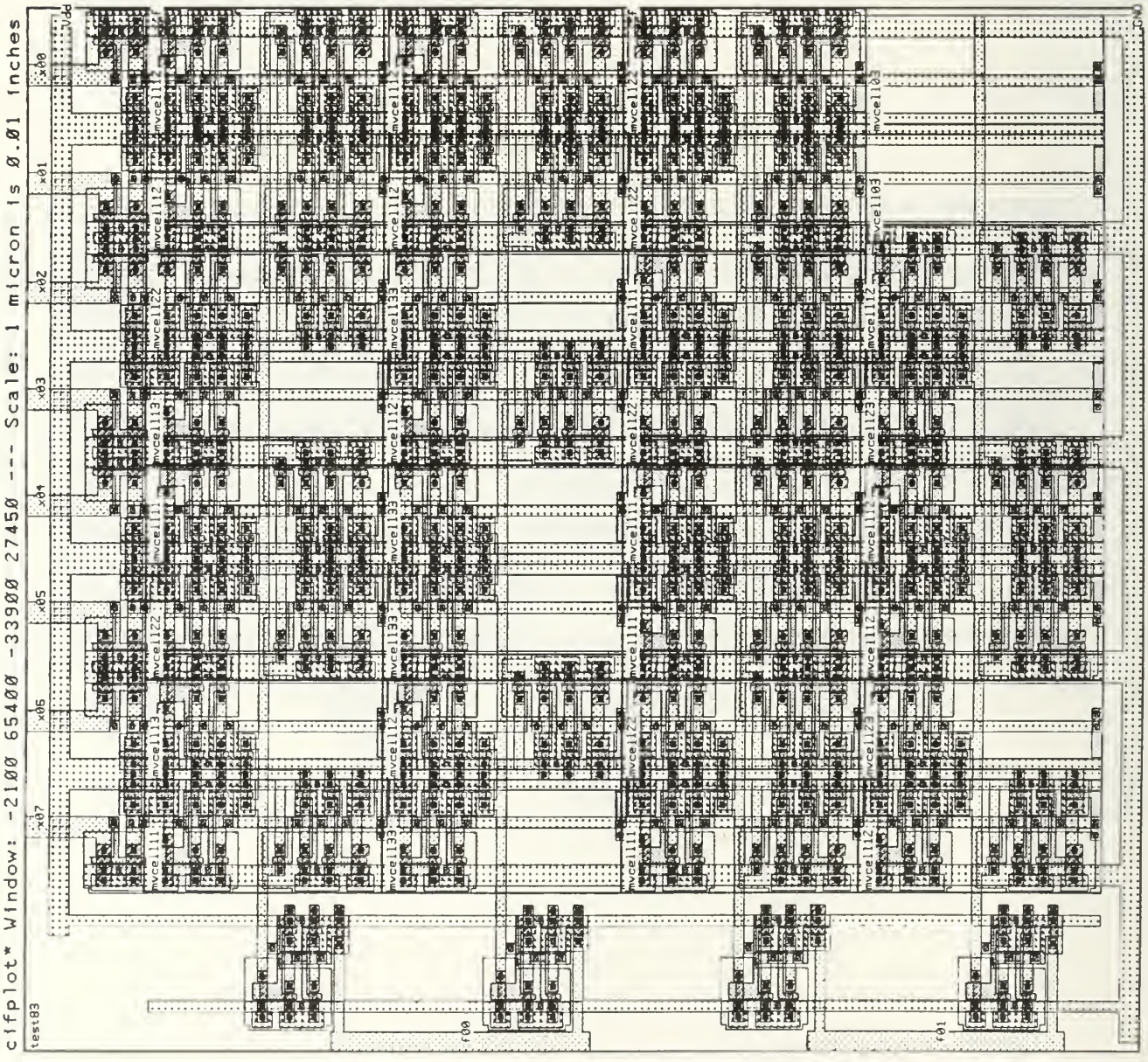
APPENDIX H

CIRCUIT LAYOUT FOR 4-INPUT 2-OUTPUT MVL FUNCTION



APPENDIX I

CIRCUIT LAYOUT FOR RANDOMLY CHOSEN MVL FUNCTION



LIST OF REFERENCES

1. Kawahito, S., and others, "A High-Speed Compact Multiplier Based on Multiple-Valued Bi-Directional Current-Mode Circuits," Proceeding of the International Symposium on Multiple-Valued Logic, pp. 172-180, May 1987.

2. Kameyama, M., Kawahito, S., and Higuchi, T., "A Multiplier Chip with Multiple-Valued Bi-Directional Current-Mode Logic Circuits," Computer, pp. 43-56, April 1988.

3. Kerkhoff, H. G., and Butler, J. T., "A Module Compiler for High-Radix CCD PLA'S", preprint.

4. Sasao, T., and Terada, H., "Multiple-Valued Logic and the Design of PLA's with Decoders," Proceeding of the International Symposium on Multiple-Valued Logic, pp. 62-70, May 1979.

5. Lee, H. S., *A CAD Tool for Current-Mode Multiple-Valued CMOS Circuits*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988.

6. Tuinenga, P. W., *SPICE A Guide to Circuit Simulation and Analysis Using PSpice*, Prentice Hall, 1988.

7. Dueck, G. W., and Miller, D. M., "A 4-Valued PLA Using the MODSUM," Proceeding of the International Symposium on Multiple-Valued Logic, pp. 45-52, May 1986.

8. Onneweer, S. P., and Kerkhoff, H. G., "High-Radix Current-Mode CMOS Circuits Based on the Truncated-Difference Operator," Proceeding of the International Symposium on Multiple-Valued Logic, pp. 131-143, May 1986.

9. *PSpice*, Microsim Corporation, 1987.

10. Scott, W. S., and others, *1986 VLSI Tools*, University of California, Berkeley, 1985.

11. Tirumalai, P., and Butler, J. T., "On the Realization of Multiple-Valued Logic Functions using CCD PLA's," Proceeding of the 14th International Symposium on Multiple-Valued Logic, pp. 32-42, May 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Prof. J. T. Butler, Code 62BU Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	3
5.	Prof. C. Yang, Code 62YA Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	2
6.	Air Force Central Library Sindaebang Dong, Gwanak Gu, Seoul, Republic of Korea	2
7.	Library of Air Force Academy Cheongwon Gun, Chung Cheong Buk Do 370-72 Republic of Korea	2
8.	Ko, Yong Ha 852, Pyoseon-Ri, Pyoseon-Myeon, Cheju Do 590-44 Republic of Korea	2
9.	Kerkhoff, Hans G. IC Technology and Electronics Group Department of Electrical Engineering, University of Twente 7500 AE Enschede, The Netherlands	1

10 Dr. George Abraham, Code 1005 1
Office of Research and Technology, Applied Physics
Naval Research Laboratories
4555 Overlook Ave, N.W
Washington, DC 20375

11 Dr. Robert Williams 1
Naval Air development Center, Code 5005
Warminster, PA 18974-5000

12 Dr. Parthasarathy Tirumalai 1
Hewlett-Packard Co.
5301 Stevens Creek Blvd, 52L/57
Santa Clara, CA 95052

13 Dr. Joo-Kang Lee 1
POSTECH Research Institute of Science and Technology
P.O. Box 125 Pohang City, Kyungbuk 680
Republic of Korea

14 LCDR John Yurchak 1
Dept. of Computer Science
Naval Postgraduate School
Monterey, CA 93943

Thesis
K716255 Ko
c.1 Design of multi-
ple-valued programmable
logic arrays.